UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

OPTIMIZATION OF PARAMETERS FOR BINARY GENETIC ALGORITHMS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

PEDRO A. DIAZ-GOMEZ

Norman, Oklahoma

2007

OPTIMIZATION OF PARAMETERS FOR
BINARY GENETIC ALGORITHMS

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

By

Dedication

This dissertation is dedicated to the memory of my parents; to my wife Emiliana, my sons and daughters Pedro Arturo, Ana Maria Carolina and David; to all my family, specially Amelio, Purita and Fanny Esperanza, my parents in law, my brothers in law; Dr. Dean F. Hougen, Dr. John K. Antonio, Dr. Millie Audas, Emily Smith, Yoana Walschap and my Professors at The University of Oklahoma.

Acknowledgments

I would like to thank Dr. Dean Hougen, for his teaching, support and patience; Dr. John Antonio, Dr. Amy McGovern, Dr. Andrew Fagg, Dr. Monte Tull, and Dr. Linda DeBrunner for being part of my committee; Dr. Sudarshan Dhall, Dr. Henry Neeman, Dr. Sesh Commuri, and all my professors at OU, from whom I learned not only in the classroom, but with example, the way to do science and be better. Thanks a lot.

Contents

List of Tables

List of Figures

Abstract

Genetic Algorithms (GAs) belong to the field of *evolutionary computation* which is inspired by biological evolution. From an engineering perspective, a GA is an heuristic tool that can approximately solve problems in which the search space is huge in the sense that an exhaustive search is not tractable. The appeal of GAs is that they can be parallelized and can give us "good" solutions to hard problems.

In the GA framework, a *species* or *population* is a collection of individuals or chromosomes, usually initially generated randomly. A predefined fitness function guides selection while operators like crossover and mutation are used probabilistically in order to emulate reproduction.

One of the difficulties in working with GAs is choosing the *parameters*—the population size, the crossover and mutation probabilities, the number of generations, the selection mechanism, the fitness function—appropriate to solve a particular problem. Besides the difficulty of the application problem to be solved, an additional difficulty arises because the quality of the solution found, or the sum total of computational resources required to find it, depends on the selection of the parameters of the GA; that is, finding a correct fitness function and appropriate operators and other parameters to solve a problem with GAs is itself a difficult problem. The contributions of this dissertation, then, are: to show that there is not a linear correlation between diversity in the initial population and the performance of GAs; to show that fitness functions that use information from the problem itself are better than fitness functions that need external tuning; and to propose a relationship between selection pressure and the probabilities of crossover and mutation that improve the performance of GAs in the context of of two extreme schema: small schema, where the building block in consideration is small (each bit individually can be considered as part of the general solution), and long schema, where the building block in consideration is long (a set of interrelated bits conform part of the general solution).

The Dissertation proposes three general hypotheses. The first one, in an attempt to measure the impact of the input over the output, study that there is not a linear correlation between diversity in the initial population and performance of GAs. The second one, proposes the use of parameters that belong to the problem itself to joint objective and constraint in fitness functions, and the third one use Holland's Schema Theorem for finding an interrelation between selection pressure and the probabilities of crossover and mutation that, if obeyed, is expected to result in better performance of the GA in terms of the solution quality found within a given number of generations and/or the number of generations to find a solution of a given quality than if the interrelation is not obeyed.

Theoretical and practical problems like the one-max problem and the intrusion detection problem (considered as problems with small schema) and the snake-in-the-box problem (considered as a problem with long schema) are tested under the specific hypotheses of the Dissertation.

Chapter 1

Introduction

GAs take their inspiration from biological evolution as proposed by Darwin. In biological evolution, individuals from species that adapt to their environment have a chance to survive and reproduce through natural selection. Species that survive usually develop new capabilities and capacities that can be inherited by offspring, if those prove to be worthwhile, and can be maintained through generations (Bäck, 1996). In GAs, a population of individuals survive and reproduce through "artificial selection." Fittest individuals are selected to go to the next generation, or are selected to mate and possibly *cross over* to produce offspring. Little changes called mutations can occur in the genotype of individuals and those can continue through generations if they prove to be "good" in maintaining fitness (Bäck, 1996).

In biology, *organisms* are composed of cells, *cells* contain chromosomes, *chromosomes* are composed of genes, and *genes* are the initial pieces for information of life (Mitchell, 1998). A gene encodes a trait, and each possible value of a gene is called an *allele* (Mitchell, 1998). The set of genetic material of a species is called its *genome*. An individual's set of genes is called its *genotype* and the *phenotype* is the expression of the genotype in an environment. In GAs, artificial genomes are evaluated "judging phenotypical expressions of genotypes" (Bäck, 1996). Likewise, in the GA framework, a *species* or *population* is a collection of *individuals* or *chromosomes*, usually initially generated randomly. The fittest are usually selected to crossover and mutate to give rise to possibly new genotypes that are sometimes better than their parents; they constitute a new *generation*. This artificial evolutionary process gives the idea of evolution to the best, i.e., climbing to a maximum.

The environment (*E*), the structure of life[1] (*A*) which is selected because it is the fittest in the environment, i.e., the one that evolves according to an adaptive plan[2] (τ), and the measure used in performing selection (μ) constitute the salient features of adaptation (Holland, 1992). There could be different environments, so a structure could be evaluated differently depending on the environments it is in. The evaluation of the structure is then a function that depends on the environment and that can, in a general form, be defined as: $\mu_E : \mathcal{A} \rightarrow \Re$ where $\mathcal{A}$ is the set of structures that are evaluated[3] in the corresponding environment $E \in \varepsilon$.

1.1 The Problem: Improving the Performance of Genetic Algorithms

This problem constitutes the core of this dissertation: How can a GA be made "robust" so that if

1. the set of all possible structures A, is large and complex, there is a method or measure that could help to choose the initial structures,

2. the function that evaluates the structure A in the corresponding environment E has multiple parameters that conflict with each other, then one knows how to choose those parameters using information from the problem domain itself, avoiding possible tuning of external parameters,

3. the operators Ω influence the evolutionary process, those influences should be understood, taking into account not only their influence in finding good solutions but also their possible influence in poor performance of the algorithm?

These points can be summarized in the following research questions:

[1]An artificial chromosome for a GA.

[2]This dissertation follows the original name given by Holland (1992), however, it would be better to call such plans "procedures."

[3]As μ_E is an evaluation function, its mapping usually goes to a real number and the range could depend on the specific environment. In some GAs it is common to consider for example $\mu_E : \mathcal{A} \rightarrow \Re^+ \cup \{0\}$ or $\mu_E : \mathcal{A} \rightarrow [0, 1]$ if μ_E is a normalized function greater than or equal to zero.

1. The structure $A \in \mathcal{A}$: Is there a correlation between *diversity* in the initial population and the performance of the algorithm, measured in the quality of solutions formed and/or number of generations to find "good" solutions?

2. The parameters that join the objective and the constraint in the function that evaluates the structure A: Do internal parameters (i.e., parameters that belong to the problem itself) result in better performance than external parameters (i.e., exogenous parameters that may be tuned according to the input)?

3. The operators Ω: Is there a relation between the selection, crossover, and mutation operators of the corresponding plan τ, that influences the evolutionary process, and if there is, what is that relation?

1.2 The Problem in Detail: Optimization of Parameters for Genetic Algorithms

A generation for a GA is a set of individuals or chromosomes that constitutes the set of structures under artificial adaptation at some step t. Usually in GAs, the first set of individuals is generated randomly and from there each one is generated using the selection mechanism and the set of operators Ω. The first generation then is generated at step 1, the next generation is generated at step 2, and so forth. This means that a GA uses the generation at step t to give rise to an entire set of chromosomes that constitute the next generation at step $t + 1$. A GA iterates from 1 through T generations, where T could be a fixed parameter or it could be the step where the fittest individual is found or where another stop criterion is satisfied.

In GAs, the environment E is the input to the algorithm, so there is a large set of possibilities that the algorithm should be able to handle and that constitutes the uncertainty of the environment. The structure A is the encoding of the approximate solution to the specific problem and that can be large and complex. Usually with binary GAs, the possible solution is encoded as $1's$ and $0's$, so the search space is $O(2^k)$, where k is the length of the structure A. The corresponding plan τ is the GA itself that uses the fitness function (performance measure) μ_E to perform selection; once the fittest structures have been selected to reproduce probabilistically, the crossover and mutation operators (Ω) are applied to them to give rise to new offspring that are expected to be better than their parents at least occasionally. This process of recombination of different structures, that gives as a result a fitter structure than the previous ones, becomes a process of climbing to a maximum in a search space (Holland, 1992; Bäck, 1996).

A typical adaptive plan followed by a GA is (Mitchell, 1998):

```
(1) Randomly generate the first population of chromosomes
(2) **Do**
    (3) Evaluate each chromosome according to the fitness function
    (4) **Iterate**
            (4.1) Probabilistically select a pair of the best evaluated chromosomes
            (4.2) Probabilistic reproduce with crossover and mutation
    **Until** next generation is completed
**Until** termination criterion has been satisfied
(5) Return the best chromosome of the population according to the fitness function
(this is the approximate solution).
```

When the problem is encoded using GAs, usually some questions arise regarding the number of individuals in the population (i.e., the size of the set of possible solutions or structures A to be tested), the probability of crossover, the probability of mutation, the number of generations to be examined, the stopping criteria, the selection pressure, and the fitness function to be used in order to solve the problem.

The number of individuals in the population influences the quality of the best solution found and/or the number of generations to find a possible solution (Lobo and Lima, 2005; Piszcz and Soule, 2006a). If the number of individuals is not large enough, then it could be that an optimum can not be reached (Jaroslaw Arabas and Mulawka, 1995; Frederick et al., 1993; Koumousis and Katsaras, 2006; Pelikan et al., 2000; Piszcz and Soule, 2006a) and, if the number of individuals is quite large then the GA could expend more computational time in finding a possible solution (Jaroslaw Arabas and Mulawka, 1995; Harik and Lobo, 1999; Koumousis and Katsaras, 2006; Lobo and Goldberg, 2004; Lobo and Lima, 2005). But, the initial population is related to diversity, in the sense that, it is expected that the more individuals a population has, the higher diversity measure is expected, and the fewer individuals a population has the less diversity measure is expected. Diversity, then, influences the quality of the solution and it is a common belief that a higher diversity can help GAs to find better solutions (Jaroslaw Arabas and Mulawka, 1995; Burke et al., 2004; Lobo and Lima, 2005; Grefenstette, 1986; McPhee and Hopper, 1999; Rosca, 1995). So, if the initial population is generated randomly, it would be useful to measure how different individuals are, if in general, this common belief is true. One contribution of this dissertation, then, is to show that there is no direct correlation between diversity and GA's performance, at least with the standard range of diversity measures used in a randomly generated population.

The fitness function μ_E is responsible for guiding evolution; it needs to guide the GA to find a "good" solution to the problem. The fitness function is at the core of the evolutionary process and it joins objectives and constraints. There are at least two forces: one that tries to maximize objectives while the other tries to minimize violations of constraints (Coello, 1998; Mé, 1998). Sometimes what happens is that the parameters that are joining objectives with constraints are tuned for specific data but when the input changes—when the environment changes—it is possible that the parameters previously found are not optimal. So, could fitness functions be built using information about the problem itself in order to set those parameters? This dissertation is going to present a hypothesis that shows the possibility of building such functions, showing a fitness function that uses only information from the problem itself that outperforms a fitness function that needs external tuning for doing misuse detection.

Looking at GA operators Ω, it can be seen that one purpose of the crossover operator is to "recombine good blocks" of two chromosomes in order to climb to a maximum according to the fitness function given. Researchers often use a probability of 70% for performing crossover (Mitchell, 1998), but why? Is this probability "good" for almost all GAs? With regard to the mutation operator, it gives the possibility of moving a chromosome to a new region of the search space; therefore, mutation helps with diversity and in moving the algorithm from stationary points. For example, if there is a specific gene which has the same allele for all individuals in a population, then in performing crossover that specific gene is never going to change. Mutation, then, is the tool that can probabilistically change that gene to a different allele. How could the probability value that should be used for mutation be determined?. The fitness function, as a performance measure, is used to perform the selection. The selection mechanism is the one that is used to select chromosomes in the population in order to reproduce (Mitchell, 1998). If the selection pressure is quite high, i.e., if the best ones are almost the only ones that are selected, then it is possible that better regions of the search space are going to be unexplored. In contrast, if the selection pressure is quite low, it is possible that the algorithm is going to spend more generations to converge. This trade-off is called exploitation vs. exploration (Mitchell, 1998). Is there a relation between the crossover and mutation probabilities and selection pressure that can help some GAs to find better solutions? This dissertation pretend to contribute in this topic showing a relationship, based on the schema theorem (see Sections 2.3.1 and 2.3.2), between selection pressure and the probabilities of crossover and mutation, that could help GAs in finding better solutions quality more quickly.

All the parameters just stated affect the performance of a GA and actually, they are inter-related, i.e., they form a system. The goal of this dissertation, then, is to gain some insight into the difficult problem of finding a good fitness function and good parameters and operators—population diversity, selection pressure, and crossover and mutation probabilities—in order to answer some questions addressed here in the context of the one-max problem and two "real world" problems: the intrusion detection problem and the snake-in-the-box problem; and to contribute to the evolutionary computation field, trying to generalize the results to a larger class of GA problems. However, it should be taken into account that GAs are so complex that there is no general rule or conclusion that can be applied to the all problems (Zitzler et al., 2000).

The number of generations has a strong influence on the GA's effectiveness in finding an approximate solution and this parameter is usually addressed experimentally. If the algorithm stops quite soon then it is likely that a good solution has not yet been found. But what, then, is an optimal number of generations, if one exists? How is this parameter related to other parameters, if there is a relation? For example, is there a relationship between the number of generations and the number of individuals in the population? A maximum on the number of generations certainly provides a stopping criterion but could the algorithm stop before that number of generations is reached? What, then, are stopping criteria that can help to improve GAs?

Most of the empirical studies realized use tournament selection, where the GA chooses a number of individuals in the tournament and from them they selects the best, and one point crossover. With tournament selection the selection pressure is more even all along all the generations than with proportional selection, where the selection pressure usually changes in each generation because the selection pressure depends on the relative fitness values of each individual. Besides that, all tests corresponds to *generational* GAs, however it is possible that some theoretical results apply to *non-generational* GAs.

Chapter 2

Foundations of Genetic Algorithms

As stated in Chapter 1, the GA framework can be enumerated in terms of the environment E, the adaptation plan τ that acts on the set of structures $\mathcal{A}$, and the environment's response μ_E to the structure. All constrain a system that iterates to an adapted structure A or set of adapted structures for an environment E.

One can think of this as an iterative process and there for can define the structure $\mathcal{A}(t) \in \mathcal{A}$ at a specific time step[1] t. $\mathcal{A}(t)$ is, then, the end result of the application of the operators Ω, from the adaptive plan τ, to selected structures from the set $\mathcal{A}$ to form the sequence $\mathcal{A}(1)$, $\mathcal{A}(2)$, through $\mathcal{A}(t-1)$. So $\mathcal{A}(t)$, besides being the structure at time step t, can be seen as a state. The path of structures $\mathcal{A}(1)$, $\mathcal{A}(2)$, through $\mathcal{A}(t)$ are generated according to the feedback (payoff) $\lambda(t) = \mu_E(\mathcal{A}(t))$ received from the environment E. The sequence $\lambda(1)$, $\lambda(2)$, ..., $\lambda(t-1)$ provides the tools for the definition of a plan τ (Holland, 1992):

$$\tau_d : \lambda \times \mathcal{A} \rightarrow \mathcal{A}.$$

If the adaptive plan τ is selecting structures stochastically, τ can be seen as a function over a probability distribution $\mathcal{P}$ of possible structures $\mathcal{A}$ (Holland, 1992), i.e., τ receives the information from the environment $\lambda(t-1)$ regarding the structure $\mathcal{A}(t-1)$ and assigns probabilities to a range of structures to select $\mathcal{A}(t)$:

$$\tau_s : \lambda \times \mathcal{A} \rightarrow \mathcal{P}.$$

where $\mathcal{P}(\mathcal{A}, t)$ is the probability to select $\mathcal{A}(t)$, and $0 \leq \mathcal{P} \leq 1$.

[1]Section 1.2 refers to step t or generation t, however here the term *time step* is used to follow Holland's terminology.

The probability distribution of the plan τ can be associated with the probabilities embedded in the set of operators Ω because $\mathcal{A}(t)$ is generated applying operators from Ω to the structure $\mathcal{A}(t-1)$ (Holland, 1992).

Following Holland (1992), this format is used for the benefit of having the chance to look at different plans $\tau \in \mathcal{J}$, see the *efficiency* χ of each one, and compare them. Efficiency can be defined in terms of how rapidly a plan accumulates payoff and the time (number of steps) it spends in finding the fittest structures. However, a particular definition of χ depends on the environment and the researcher creating the algorithm. A useful way to consider χ is: if the plan τ is deterministic then $\tau_d : \lambda \times \mathcal{A} \rightarrow \mathcal{A}$, and the *cumulative payoff* for τ_d at time step T is (Holland, 1992)

$$U_{\tau_d,E}(T) = \sum_{t=1}^{T} \mu_E(\mathcal{A}_{\tau_d}(t))$$

where $\mu_E(\mathcal{A}_{\tau_d}(t))$ is the payoff (environment's response) for the structure $\mathcal{A}$, in the environment E, at time t. If the plan τ is stochastic then $\tau_s : \lambda \times \mathcal{A} \rightarrow \mathcal{P}$ and $\mu_E(\mathcal{A}_{\tau_s}(t))$ is replaced by the *expected payoff* $\overline{\mu}_E(\tau_s, t)$ in order to obtain

$$U_{\tau_s,E}(T) = \sum_{t=1}^{T} \overline{\mu}_E(\tau_s, t)$$

where $\overline{\mu}_E(\tau_s, t) = \sum_j \mathcal{P}(\mathcal{A}_j, t)\mu_E(\mathcal{A}_j)$, with $\mathcal{A}_j \in \mathcal{A}$ being the structure selected with probability $\mathcal{P}(\mathcal{A}_j, t)$. With this in mind, Holland (1992) suggests the following as a bound for comparing efficiency χ of plans τ_s: In the first T steps the greatest possible cumulative payoff is:

$$U_E^*(T) \overset{lub}{=}_{\tau_s \in \mathcal{J}} U_{\tau_s,E}(T) \tag{2.1}$$

where lub is the least upper bound, and

$$\lim_{T \rightarrow \infty} \frac{U_{\tau_s,E}(T)}{U_E^*(T)} = 1 \tag{2.2}$$

which means that in the limit the rate at which τ accumulates payoff is the same as the best possible rate (Holland, 1992). However, there is the problem of $T \rightarrow \infty$, so Holland

(1992) suggests a lower bound using a non-negative decreasing succession $< s_T >$ such that

$$\frac{U_{\tau,E}(T)}{U^*_E(T)} > (1 - < s_T >) \tag{2.3}$$

for all T and where $< s_T >$ could be something like $< 1/T^k >$ with $k > 0$.

Summarizing, a problem in adaptation is well posed (Holland, 1992) once $\tau \in \mathcal{J}$, $E \in \varepsilon$, and χ have been specified along with the set $(\mathcal{A}, \Omega, \lambda, \tau)$, where
$\mathcal{A} = \{A_1, A_2, ...\}$ is the set of structures, i.e., the domain of the adaptive plan,
$\Omega = \{\omega_1, \omega_2, ...\}$ is the set of operators for modifying structures,
λ is the feedback received from the environment, and
τ is the adaptive plan that determines the operator to be used for the structure $\mathcal{A}(t)$.

2.1 The Schema Theorem

As stated at the beginning of this chapter, the goal of GA research in general is to find an algorithm that improves the performance of $\mathcal{A}(t)$ according to plan τ in the environment E over time. If the algorithm is at time step T, then a cumulative average measure of the performance of the structure $\mathcal{A}(t)$ gained from $t = 1, 2, ...T$ is

$$\overline{\mu}_E(T) = \frac{1}{T}\sum_{t=1}^{T} \mu_E(\mathcal{A}_{\tau_d}(t)). \tag{2.4}$$

At time step T the ideal is to find structures ξ such that $\mu_{\xi,E} > \overline{\mu}_E(T)$. In order to identify a subset of structures, Holland (1992) gives the general concept of *schemata*,[2] which are subsets of structures taken from the power set $P(\mathcal{A}(t))$. To do that, a new symbol $*$ is introduced (Mitchell, 1998) that indicates any allowable allele value.[3]

The interpretation of schema is particular to each problem. For example, if we take a representation of a 4-dimensional hypercube (see Figure 2.1), where each node is labeled in binary notation, the schema $0{*}{*}{*}$ denotes the lower hypercube of dimension 3, i.e., the subset of points 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111. Likewise, $1{*}{*}{*}$ denotes

[2]Mitchell (1998) indicates the term either as schemata or schema.

[3]0 or 1 for binary GAs.

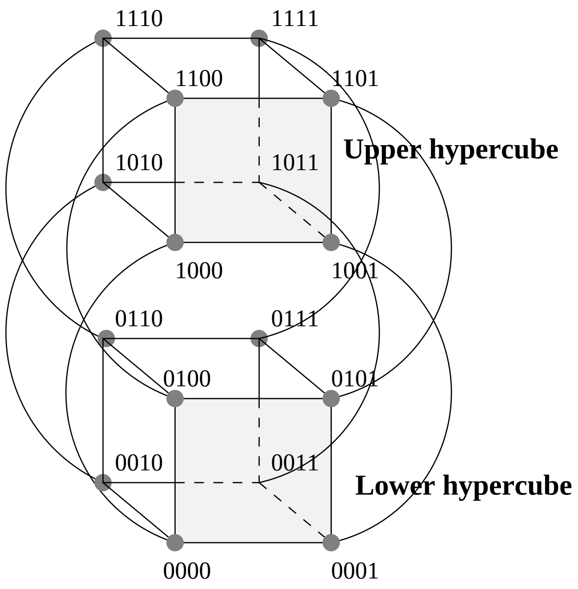


Figure 2.1: Schema in a 4-dimensional hypercube.

the upper hypercube of the same dimension. The schema $**0*$ denotes the front plane of the hypercube, i.e., the subset of points 0000, 0001, 0100, 0101, 1000, 1001, 1100, 1101.

Now the questions are, what are the schema ξ that increase the performance of $\mathcal{A}$ at time step $T+1$ and how many of those schema are there? To solve these questions, Holland (1992) proposes the following algorithm:

1. Find schema ξ such that $\hat{\mu}_{\xi,E} > \overline{\mu}_E(T)$, where $\hat{\mu}_{\xi,E}$ is the sample average performance of schema ξ in environment E; if there is no such schema ξ, stop.
2. Generate *new*[4] instances of the above-average schema ξ, returning from time to time to step 1 to locate new schema ξ' such that $\hat{\mu}_{\xi',E} > \overline{\mu}_E(T)$, particularly if $\overline{\mu}_E(T) \approx \hat{\mu}_{\xi,E}$.

This type of search, as pointed out by Holland (1992), intuitively is an improvement over a random search or an enumerative one because the algorithm is searching for instances $\xi \in \mathcal{A}$ such that it continuously increases the previous average payoff of structure $\mathcal{A}(t)$.

But there is still a question to answer: How many schema ξ are needed in order to follow the previous algorithm? Holland (1992) addresses here the encoding of the structure

[4]The fact that the instances are to be new is emphasized by Holland (1992).

$\mathcal{A}$, in the sense that the longer the structure is, the more schema there are. For example, the nodes of the 4-dimensional hypercube in Figure 2.1 are encoded in binary using a structure of length 4, which gives 3^4 schema.[5] If a hexadecimal notation was used to encode the nodes, the structure would have length 1 and 17^1 schema (Seventeen symbols to use in one position). For reproductive plans, then, longer structures with fewer allele values are better than shorter length structures with more allele values because the more schema there are, the more possibilities there are from which to choose (Holland, 1992).

Formally, the set of all possible schema is $\Xi = \{V_i \bigcup \{*\}\}^l$, where $V_i = \{0, 1\}$ for binary GAs and l is the length of the structure. Following the previous algorithm, schema ξ are sampled according to the feedback $\mu_{\xi,E}$. So, schema ξ can be seen as an event in the space $\mathcal{A}$. But, how is $\mu_{\xi,E}$ calculated for schema ξ? It can be approximated as an average of the number of instances of schema ξ sampled. For example, for the schema $\xi = 0{*}0{*}$ and the samples 0000, 0001, and 0101, the sample average is $\hat{\mu}_{0*0*,E} = (\mu_E(0000) + \mu_E(0001) + \mu_E(0101))/3$. This is the *sample average performance* $\hat{\mu}_{\xi,E}$ of the first step of Holland's algorithm, above.

2.2 Selection Mechanism

Looking at how adaptive plans τ work, the structure $A(t) \in \mathcal{A}$ must be selected in order to undergo the corresponding operators Ω, giving rise, possibly, to new offspring. The *selection mechanism*, then, gives the pressure toward the fittest individuals (Bäck, 1996) using as a measure the payoff $\mu_E(A)$ (Holland, 1992).

Various selection mechanisms have been proposed, like proportional probability selection and tournament selection (Mitchell, 1998). However, the drawback of any selection mechanism is the balance between exploitation and exploration. If the selection mechanism makes extensive use of exploitation, it can get trapped in a local minimum. If, on the contrary, the selection mechanism makes extensive use of exploration, then it may take a longer time to converge or it may not converge at all (Bäck, 1996).

The selection mechanism that perhaps follows Holland's ideas in the closest way is proportional probability selection. In *proportional probability selection* an individual is evaluated and the probability to be selected is proportional to its fitness value $\mu_E(A)$. That

[5] Three symbols used in four positions.

is, if the population at time t is $\mathcal{B}(t) \subset \mathcal{A}$, then structure $A \in \mathcal{B}(t)$ can be selected with a probability equal to $\mu_E(A)/\sum \mu_E(B_i)$ for all $B_i(t) \in \mathcal{B}(t)$ (Bäck, 1996). An alternative is q-tournament selection. In q-tournament selection, q individuals $B_i(t) \in \mathcal{B}(t)$ are chosen randomly, each with equal probability, and the one that has the greatest fitness value $\mu_E(B_i), for 1 \leq i \leq q$ among those q is the one that finally is selected.

In this dissertation most of the experiments use q-tournament selection because of the ability to control selection pressure. With proportional probability selection, as the algorithm iterates it is expected that the selection pressure decreases because the algorithm is looking for the fittest individuals which begin to dominate. With q-tournament selection, the selection pressure is even over all iterations, with the possibility that if an algorithm needs a higher or lower selection pressure, then q can be increased or decreased accordingly.

2.3 Genetic Operators

As stated at the beginning of Chapter 2, the evolutionary process of a structure $\mathcal{A}$ occurs as an application of operators Ω to $\mathcal{A}$, according to the adaptive plan τ. That is, the sequence $\mathcal{A}(1)$, $\mathcal{A}(2)$, through $\mathcal{A}(T)$ evolves as a consequence of adaptation of structure $\mathcal{A}$ to the environment *E*, according to plan τ.

Section 1.2 delineated an adaptive plan τ, which is restated here to highlight the use of operators (Holland, 1992):

```
(1) Set $t = 0$ and randomly initialize $\mathcal{B}(t) \subset \mathcal{A}$
(2) Calculate and store $\mu_E(B_i)$ for all $B_i \in \mathcal{B}(t)$
(3) Increment $t$
**Iterate**
    (4) Select $B_1 \in \mathcal{B}(t-1)$ based on proportion of better $\mu_E(B_i(t))$ found in step (2)
    (5) Select $B_2 \in \mathcal{B}(t-1)$ based on proportion of better $\mu_E(B_i(t))$ found in step (2)
    (6) Determine randomly an operator $\omega_t \in \Omega$ to be applied to $B_1$ and/or $B_2$
    (7) Use $\omega_t$ to produce a two new structures $A'_1$ and $A'_2$
    (8) Store $A's$ in $\mathcal{B}(t)$
**Until** next generation is completed
If termination criterion is not satisfied go to step (2)
```

(9) Return the best chromosome of the population according to the fitness function

The set of operators Ω acts on structures in $\mathcal{B}$. The *crossover* operator recombines structures, exploiting information already obtained and trying to explore new areas of the search space. The *mutation* operator rescues the algorithm from the possibility of losing a specific allele in $\mathcal{B}$.[6]

2.3.1 Crossover

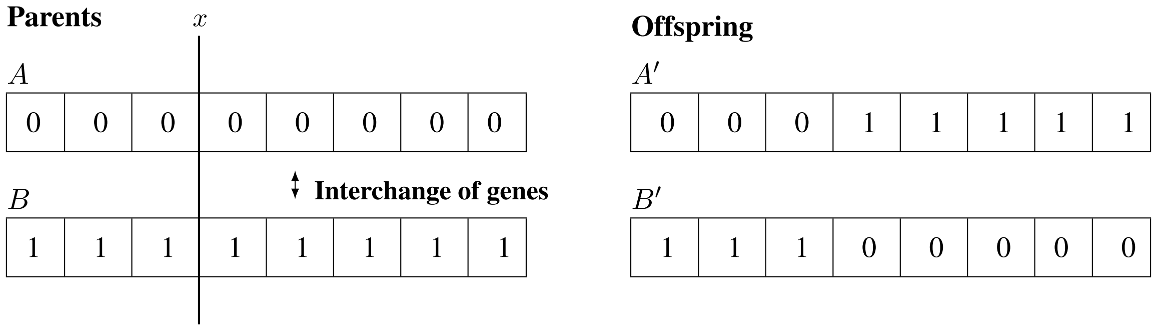


Figure 2.2: A and $B \in \mathcal{B}$ crossing over to generate A' and B'.

The crossover operator is applied to two individuals in the population in order to give raise to two offspring. A random number $1 \leq x \leq l$, with l the length of the cromosome, is choosen. The first offspring is generated with the first part (bits 1 to x) from the first parent and the last part (bits ($x + 1$ to l) from the second parent. The second offspring is generated with the first part (bits 1 to x) from the second parent and the last bits ($x + 1$ to l) from the first parent. Figure 2.2 shows the way the one point crossover operator acts on two structures A and B from $\mathcal{B}$. A point $1 \leq x < l$, where l is the length of the chromosome, is chosen at random. Two new structures A' and B' are generated. The first one, A', takes the first x alleles from A and alleles from $x + 1$ to l from B; the second one B', takes the first x alleles from B and from $x + 1$ to l from A.

Consider the effect of the crossover operator in a pool $\mathcal{B}(0)$ as given in Table 2.1, where some schema have been chosen to show the effect of the crossover operator on

[6]Holland (1992) also suggests the *inversion* operator in order to try to explore new schema in the sense that some may not be present in $\mathcal{B}$. However, inversion is usually not used by the GA community.

#	*Chromosome*																*Schema*																*Class*
1	1	1	0	1	1	0	1	0	1	1	0	0	0	1	1	1	1	1	$*$	$*$	1	0	$*$	0	$*$	$*$	$*$	$*$	$*$	$*$	1	$*$	S_1
2	1	0	1	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	$*$	0	$*$	$*$	$*$	$*$	0	1	0	$*$	$*$	$*$	1	1	S_2
3	0	0	1	0	0	0	1	1	1	1	0	1	1	1	1	1	0	0	$*$	0	$*$	0	1	$*$	1	$*$	$*$	1	$*$	$*$	$*$	$*$	S_3
4	1	0	1	0	1	1	1	1	0	1	0	0	1	0	1	1	1	0	$*$	0	$*$	$*$	$*$	$*$	0	1	0	$*$	$*$	$*$	1	1	S_2
5	1	0	0	0	0	0	0	1	0	1	0	1	0	1	1	1	1	0	$*$	0	$*$	$*$	$*$	$*$	0	1	0	$*$	$*$	$*$	1	1	S_2
6	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	1	$*$	$*$	1	0	$*$	0	$*$	$*$	$*$	$*$	$*$	$*$	1	$*$	S_1
7	1	1	1	1	1	0	1	0	1	1	0	1	0	1	1	0	1	1	$*$	$*$	1	0	$*$	0	$*$	$*$	$*$	$*$	$*$	$*$	1	$*$	S_1
8	1	1	1	0	1	0	0	0	1	0	1	1	1	0	1	1	1	1	$*$	$*$	1	0	$*$	0	$*$	$*$	$*$	$*$	$*$	$*$	1	$*$	S_1
9	0	0	0	0	1	0	1	1	1	1	1	1	0	0	0	1	0	0	$*$	0	$*$	0	1	$*$	1	$*$	$*$	1	$*$	$*$	$*$	$*$	S_3
10	0	0	0	0	0	0	1	0	1	0	1	1	1	1	0	0	0	0	$*$	0	$*$	0	1	$*$	1	$*$	$*$	1	$*$	$*$	$*$	$*$	S_3

Table 2.1: Initial set $\mathcal{B}(0)$ of 10 chromosomes and some schema.

#	*Chromosome*																*Schema*																*Class*
1	0	0	1	0	0	0	1	1	1	1	0	1	1	1	1	1	0	0	$*$	0	$*$	0	1	$*$	1	$*$	$*$	1	$*$	$*$	$*$	$*$	S_3
2	1	1	1	1	1	0	1	0	1	1	0	1	0	1	1	0	1	1	$*$	$*$	1	0	$*$	0	$*$	$*$	$*$	$*$	$*$	$*$	1	$*$	S_1
3	1	0	1	0	1	1	1	1	0	1	0	1	0	1	1	0	1	0	$*$	0	$*$	$*$	$*$	$*$	0	1	0	$*$	$*$	$*$	1	0	S_4
4	1	1	1	1	1	0	1	0	1	1	0	0	1	0	1	1	1	1	$*$	$*$	1	0	$*$	0	$*$	$*$	$*$	$*$	$*$	$*$	1	$*$	S_1
5	0	0	0	0	1	0	1	1	1	1	0	1	1	1	1	1	0	0	$*$	0	$*$	0	1	$*$	1	$*$	$*$	1	$*$	$*$	$*$	$*$	S_3
6	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	1	0	0	$*$	0	$*$	0	1	$*$	1	$*$	$*$	1	$*$	$*$	$*$	$*$	S_3
7	1	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	$*$	0	$*$	$*$	$*$	$*$	0	1	0	$*$	$*$	$*$	0	0	S_5
8	0	0	0	0	0	0	1	0	1	0	1	1	1	1	1	1	0	0	$*$	0	$*$	0	1	$*$	1	$*$	$*$	1	$*$	$*$	$*$	$*$	S_3
9	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	1	$*$	$*$	1	0	$*$	0	$*$	$*$	$*$	$*$	$*$	$*$	1	$*$	S_1
10	1	0	1	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	$*$	0	$*$	$*$	$*$	$*$	0	1	0	$*$	$*$	$*$	1	1	S_2

Table 2.2: Set $\mathcal{B}(1)$ of chromosomes after crossing over some pairs of chromosomes from set $\mathcal{B}(0)$. Two new schema S_4 and S_5 have emerged.

them.[7]. At $t = 1$ the effect of crossover using a *probability of crossover* $p_c = 60\%$ is shown in Table 2.2. Structures number 3 and 7 are selected from $\mathcal{B}(0)$ with a probability of 60% that they are going to be crossed over. In this case they did not cross over, so they are copied to the next generation $\mathcal{B}(1)$ in positions 1 and 2. Now, individuals 4 and 7, belonging to schemas S_2 and S_1, respectively, are selected from $\mathcal{B}(0)$. This time, they crossover at position $x = 11$ to give rise to structure 1010111101010110 (which is an example of a new schema $10{*}0{*}{*}{*}{*}010{*}{*}{*}10$) and structure 1111101011001011 (which belongs to schema S_1 as does its parent 7). The process continues until the entire set $\mathcal{B}(1)$ is generated.

[7]Other schema could be chosen (for example, $111{*}{*}0{*}0{*}{*}{*}{*}{*}{*}1{*}$) However, for simplicity these were the ones chosen. For this case there are 3^{16} schema, and just 3 of those are exemplified.

If the *rank* of schema (Holland, 1992) is defined as the count of schema in pools $\mathcal{B}(0)$ and $\mathcal{B}(1)$, it can be appreciated that in $\mathcal{B}(0)$, S_1 is given a rank of 4, S_2 is given a rank of 3, and S_3 is given a rank of 3. In $\mathcal{B}(1)$, S_1 is given a rank of 3, S_2 is given a rank of 1, S_3 is given a rank of 4, S_4 is given a rank of 1, and S_5 is given a rank of 1. So the rank changed.[8] In $\mathcal{B}(0)$ the highest ranked was S_1, now in $\mathcal{B}(1)$ the most dominant is S_3 and schema S_2 loses rank and now it is competing with two new schema S_4 and S_5. So there are at least three aspects to highlight here:

1. the generation of new instances of schema that already exist in the pool—as in the case of schema S_3,
2. the generation of new schema, i.e., schema that do not belong to the set $\mathcal{B}(0)$—as with S_4 and S_5,
3. the loss of instances of schema—as with S_1 and S_2, and
4. the loss of some schema from the pool.[9]

These four aspects comprise the tasks accomplished by the crossover operator.

Looking in detail at the constructive factor, i.e., when the instances of a schema increase, for instance, structure S_3 ($00{*}0{*}01{*}1{*}{*}1{*}{*}{*}{*}$), if the crossover point is beyond $x = 12$ the schema is not going to be destroyed. However, for $1 < x < 12$ there is a possibility for the schema to be disrupted because in that range there are $h = 7$ defined positions. *Defining length* of a schema $d(\xi)$ is defined as the distance between the two extreme defined positions, i.e., the count of the positions between them (Mitchell, 1998). For example, $d(00{*}0{*}01{*}1{*}{*}1{*}{*}{*}{*}) = 12 - 1 = 11$. The bigger $d(\xi)$ the greater the possibility for schema ξ to be destroyed.

Theorem 1 (Holland, 1992) *Given $n(\xi, t)$, the number of schema ξ at step t, the* expected number *($\mathcal{N}$) of schema ξ at step $t+1$ after proportional probability selection and one point crossover is*

$$\mathcal{N}(\xi, E, t+1) \geq \left(1 - p_c \frac{d(\xi)}{l-1}(1 - p_\xi)\right) \frac{\hat{\mu}_{\xi,E}(t)}{\overline{\mu}_E(t)} n(\xi, t)$$

[8]One can differentiate *rank* from *importance* in the sense that perhaps the most important are those schema that have more defined bits—as schema S_2 in $\mathcal{B}(0)$—in the sense that those have more information in them.

[9]For this particular example there is no loss of any particular schema considered.

where p_c is the probability of crossover, p_ξ is the probability of crossing with the same schema ξ, and l is the length of the structure.

Following Holland (1992) and Mitchell (1998) the explanation of terms is like this:

- if the number of schema ξ is $n(\xi, t)$ at step t, then the expected number of those schema at step $t+1$ is $\hat{\mu}_{\xi,E}(t)\frac{n(\xi,t)}{\overline{\mu}_E(t)}$, assuming proportional probability selection,
- if two structures are selected with probability p_c for one point crossover then, if the point x at which crossover is to take place is within the defining length $d(\xi)$ then schema ξ can be disrupted; the probability of this is $p_c\frac{d(\xi)}{l-1}$,
- if the two structures selected belong to the same schema ξ with probability p_ξ, then the effect of crossover does not change the schema, so the possible effect of disruption is $1 - p_\xi$,
- so the net chance of disruption of schema ξ is $p_c\frac{d(\xi)}{l-1}(1-p_\xi)$.

Concluding, then, the building effect of crossover at step $t+1$ is

$$1 - p_c\frac{d(\xi)}{l-1}(1-p_\xi)$$

in a proportion equal to $\hat{\mu}_{\xi,E}(t)\frac{n(\xi,t)}{\overline{\mu}_E(t)}$. The $\geq$ holds in the sense that other schema ξ' can give rise to schema ξ in doing crossover (just as schema ξ can give rise to another schema ξ'' as the net effect of one point crossover).

2.3.2 Mutation

The mutation operator is applied randomly to a bit of a chromosome, randomly choosen. When the mutation operator is applied, it flips the value of the bit. This operator is used because as the set of structures $\mathcal{A}$ evolves as a result of the selection mechanism in conjunction with the crossover operator, some alleles can be lost because some schema can disappear. That could be the case for locus (location) 6 in the population in Table 2.2 where a great percentage (90%) of individuals have allele 0. Likewise, a 0 in locus 4 of schema S_1 would give us allele 0 in all individuals at locus 4. The only way to again try an allele of value 1 in locus 4 is through the mutation operator; the crossover operator can not restore that allele.

If the *probability of mutation* per bit is p_m and the the number of defined positions is h, then the probability of not disrupting the schema ξ is $(1 - p_m)^h$ and the probability of disruption would be $1 - (1 - p_m)^h$.

In conclusion, using crossover and mutation operators, the expected number of schema ξ at time step $t + 1$ is given by the following corollary.

Corollary 1 (Holland, 1992) *Given $n(\xi, t)$, the number of schema ξ at step t, the* expected number *of schema ξ at step $t + 1$ after proportional probability selection, one point crossover, and mutation is*

$$\mathcal{N}(\xi, t+1) \geq \left(1 - p_c \frac{d(\xi)}{l - 1}(1 - p_\xi)\right)(1 - p_m)^h \, \frac{\hat{\mu}_\xi(t)}{\overline{\mu}(t)} n(\xi, t).$$

Chapter 3

State of Research of the Intrusion Detection Problem

As stated in Chapter 1, the goal of this dissertation is to work on the difficult problem of understanding what constitutes good parameters and operators for GAs in the context of two particular problems: intrusion detection and the snake-in-the-box problem. Both problems are appealing for use with GAs principally because the search space is huge. In the case of intrusion detection the search space is of magnitude 2^k, where k is the number of intrusions the algorithm is looking for. For the snake-in-the-box problem it is 2^n, where $n = 2^d$ and d is the dimension of the hypercube. However, for the present, we have choosen the intrusion detection problem because it is a real problem that give us a good example of a fitness function that tries to maximize a value satisfying a second constraint.

3.1 The Intrusion Detection Problem

The intrusion detection problem is the computer security problem of protecting the integrity, confidentiality, and availability of data from intruders (Crosbie and Spafford, 1995). An *intruder* is a "subject" (defined by Denning (1986) as an entity that performs an action over an "object") that attempts to gain access to computer resources by subverting the system or without valid authorization. Once an intruder is in the system, there is an intrusion. Besides that, a "valid user" can exceed his or her privileges in an attempt to gain confidential information. He or she is an intruder too. There are both external and internal intruders (Anderson, 1980) and, likewise, external and internal intrusions.

An intruder can exploit a system's vulnerabilities, defined as those holes in systems, like the ones found in Microsoft Internet Explorer 5.0, Netscape Enterprise Server 3.6, Real Secure Network Intrusion Detection Software, and so on (Schneier, 2000) (some statistics in this topic can be found in (Diaz-Gomez and Hougen, 2006e).

An *incident* is an attack; for example, e-mail spoofing, computer virus spreading, and so forth. Some attacks can be known in advance so there should be security mechanisms that protect the system against those. Intrusion detection systems are some of those mechanisms.

An intrusion detection system is a system that looks for penetrations in computers and computer networks (Bace, 2000). *Misuse detection* refers to the detection of internal penetrations (Tjaden, 2004), i.e., users that use the computer system gaining unauthorized privileges in order to obtain their own benefits or destroy important data. In misuse detection, penetration methods are known in advance, so the intrusion detection system is matching user activity against the known set of penetrations. *Anomaly detection* refers to the detection of abnormal activity, i.e., activity which is a deviation from normal activity (Denning, 1986). In anomaly detection, normal activity is known in advance, so that the system can compare between normal and abnormal.

Some of the difficulties with misuse detection systems are (1) that the search space can be huge, (2) that the system can give false positive and/or false negative alarms, and (3) that exclusive attacks can be present (Mé, 1998)—i.e., there may be attacks that can occur independently but not at the same time as other attacks because of violations of the constraint— If a computer system tries to look for k known intrusions directly in the audit trail file, that could be prohibitively expensive because of the huge amount of data, which makes heuristic tools appropriate for approximately solving this kind of problem (Mé, 1998). For their part, anomaly detection systems have the difficulty of differentiating between "normal" and "abnormal" activity; usually these types of systems use statistical models in order to gain information on these two classes (Denning, 1986). False positive and false negative errors are inherent problems for all intrusion detection models (Bace, 2000) and, of course, the lower the number of those errors, the better the intrusion detection system.

Intrusion detection systems can be classified according to the type of system they are monitoring as host, multi-host, or network-based (Tjaden, 2004), and whether the detection is taking place in real-time or off-line (Bace, 2000). Intrusion detection systems usually use *audit trail logs* which are composed of information gathered from the operating system (Bace, 2000).

This dissertation is going to present a GA that performs the task of an off-line misuse detection system, not only as a practical experiment that shows the benefit of using a heuristic tool in solving a particular security problem but as an example system to use in order to explore the theory behind the use of GAs.

3.2 A Genetic Algorithm Approach to Misuse Detection

One motivation for this topic was the paper "*GASSATA*, A Genetic Algorithm as an Alternative Tool for Security Audit Trail Analysis" (Mé, 1998). *GASSATA* is an off-line tool that increases security audit trail analysis efficiency. The goals of this approach are the following:

- to investigate misuse detection, i.e., to determine if the events generated by a user correspond to known attacks, and
- to search in the audit trail file for the occurrence of attacks by using a heuristic method (GAs) because this search is an NP-complete problem.

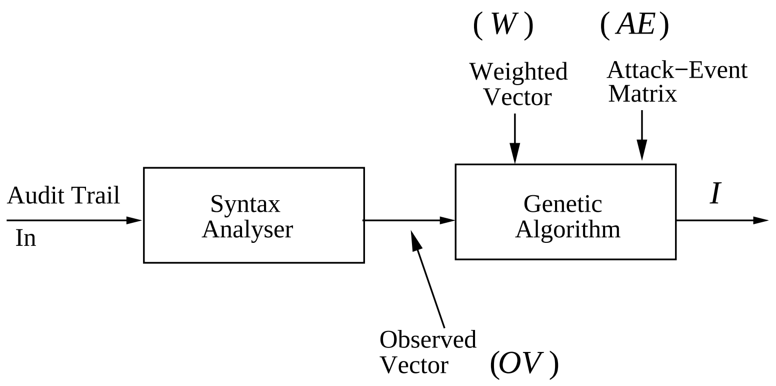


Figure 3.1: Prototype of *GASSATA*.

This approach is shown schematically in Figure 3.1. The audit subsystem recognizes various kinds of events (such as changing to a particular directory or copying a file) which are recorded in the audit trail file. The *Syntax Analyzer* classifies those audit events and generates the *Observed Vector* (OV), which is the aggregation of all the activity performed by a user during some period of time. The *GA* module finds the hypothesized vector I

that maximizes the product $W \cdot I$, subject to the constraint $(AE * I)_i \leq OV_i$ (it is noticed that W does not appears in the constraint because the constraint is that the set of intrusions must be possible, and W tell us how important an intrusion is to be, not weather it is possible or not), where W is the *weighting vector* of I that reflects the priorities of the security manager,[1] AE is the *Attack-Event* matrix that correlates sets of events with known attack profiles, $1 \leq i \leq N_e$, and N_e is the number of events. See Figure 3.2 for an example of an *Attack-Event* matrix AE which is encoded by the security manager with the type of intrusions to look for, an example of an I vector of the type that needs to be maximized (the goal is to find all possible intrusions, therefore avoiding false negatives), and an observed vector OV which tells us about the user's activity. Furthermore, Table 3.2 shows the current multiplication of the AE matrix with the hypothesis vector of possible intrusions I (this product is compared with the actual observed activity OV in order to look for violations of the constraint; if $(AE * I)_i > OV_i$, then the algorithm is hypothesizing higher occurrences of activity i than really happened, so there is a violation of the constraint) and columns T and T' which are related to constraint violations (see Sections 3.3.1 and 3.3.2 for further explanations of T and T').

3.3 Fitness Functions for Misuse Detection

Most of the success of GAs in finding a solution is attributed to the fitness function in the sense that convergence to correct solutions depends in great part on it. A fitness function that captures all goals and constraints can be straightforwardly proposed. However, setting parameters that appropriately join goals and constraints can be more difficult to find (Diaz-Gomez and Hougen, 2005c). The fitness function suggested for *GASSATA* (Mé, 1998) is defined as:

$$F(I) = \alpha + \sum_{i=1}^{N_a} W_i * I_i - \beta * T^2. \tag{3.1}$$

The goal of each component of the equation is as follows:

- α: To maintain $F(I) > 0$, and therefore maintain diversity in the population.

[1] $W_j = 1$, for all j, was used in all test performed in this dissertation.

EVENT #	ATTACK #																												
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	*I*	*AE*I*	*OV*	*T*	*T′*
0	3																								0	0	0	0	0
1					1																				0	0	0	0	0
2				1																					0	1	0	1	1
3		3																							1	0	0	0	0
4			3																						0	0	0	0	0
5		3							8																1	8	0	1	1
6						5														1		5			1	10	0	1	2
7						30																			1	30	76	0	0
8							5																		1	5	0	1	1
9										3															0	0	0	0	0
10											2														1	2	20	0	0
11												3													1	3	0	1	1
12															10	1									1	0	0	0	0
13																1									0	0	6	0	0
14																1									0	0	0	0	0
15																								4	0	4	4	0	0
16																				1					0	0	0	0	0
17		3				35	5		8	3	2	3			10	3		300		2		5		4	0	62	94	0	0
18																			100						1	100	0	1	1
19								5																	0	5	42	0	0
20														10											1	0	0	0	0
21																				1					1	0	0	0	0
22															10										0	0	0	0	0
23																						5			1	5	5	0	0
24																							1			0	0	0	0
25					1																3					3	459	0	0
26													30													30	1335	0	0
27																	50									0	0	0	0

Figure 3.2: Example of an Attack Event matrix AE, I vector to be maximized, the product of AE and I, the observed vector OV and counts of faults T and T'.

- I: To allow for hypotheses of attacks. The system is rewarded for hypothesizing attacks, particularly those of greatest concern to the security manager. This is generated randomly in the first generation.
- β: To provide a slope for the penalty function.
- T: To count the number of times the constraint is violated, i.e., $(AE * I)_i > OV_i$. The system is penalized, then, for hypothesizing sets of attacks that could not have occurred, given the observations.

Mé (1998) reported results which seam to satisfy his approach to the problem. However, our approach shows weakness in his approach.

3.3.1 Analysis of *GASSATA'S* Fitness Function

In order to appreciate the problem of finding appropriate parameters to combine objectives and constraints, we conducted the following analysis (Diaz-Gomez and Hougen, 2005c): take Equation 3.1 and, as a first approximation, choose the parameter $\beta = 1.0$, so that the penalty term is T^2. As $0 \leq T \leq N_e$ then $0 \leq T^2 \leq N_e^2$. Assuming that, on average, the count of faults is $T = N_e/2$, then the penalty term would be, on the average, in the range of $0 \leq T^2 \leq N_e^2/4$. As the initial testing was done with proportional probability selection, a positive fitness value allows for less frequent elimination of individuals—compared with negative fitness values where the individual essentially disappears from the current population—so, α was set as $\alpha = N_e^2/2$ (twice as large as the average penalty term assumed). W_i, which corresponds to the risk of each attack i, was assumed to be equal to 1 for all i so that after finding all possible intrusions, those can be sorted according to the weights given by the security manager. With these parameters and using as input a file corresponding to user 2051[2]—with 4 known intrusions—Table 3.1 shows the results. A large number of false positives and some false negatives are observed in the 11 runs performed with the same input file.

Analyzing the term $\sum_{i=1}^{N_a} W_i * I_i$ in Equation 3.1, it can be inferred that this term guides the solution to have a maximum number of intrusions, because this is the rewarding term. However, this is good only until the correct set of intrusions are found. If more intrusions than that are hypothesized, the problem of false positives occurs.

[2]The data used was downloaded from the Lincoln Laboratory at MIT (Fried and Zissman, 1998).

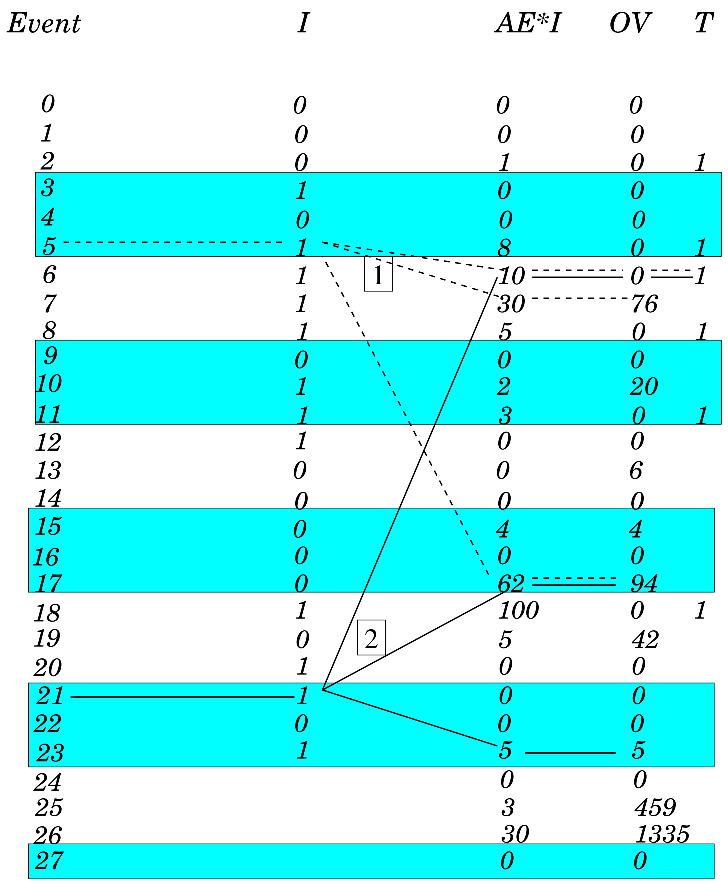
Run	False +	False –	Detected	Greatest Fitness
0	7	0	4	649.0
1	8	1	3	672.0
2	6	1	3	682.0
3	10	1	3	637.0
4	9	0	4	648.0
5	9	0	4	674.0
6	10	1	3	674.0
7	9	1	3	663.0
8	11	0	4	655.0
9	9	1	3	610.0
10	8	1	3	645.0

Table 3.1: User 2051 at 11:00 am; $\alpha = N_e^2/2$; $\beta = 1$; $500,000$ generations; 11 runs.

Similarly, the term $\beta * T^2$ decreases the fitness value but various intrusions can require the same event. When this happens, the counting of overestimates is wrong. See, for example, Figure 3.3. In $\boxed{1}$ (dashed lines in Figure 3.3) there is a first case: A type 5 intrusion was hypothesized. This intrusion requires occurrences of events of types 6, 7, and 17. For event type 6, the hypothesis gives a number of occurrences greater than the number of required occurrences that really happened; T is then incremented by 1. For event types 7 and 17 there is no penalty since $(AE * I)_7 \leq OV_7$ and $(AE * I)_{17} \leq OV_{17}$ (that is, $30 \leq 76$ and $62 \leq 94$).

In $\boxed{2}$ (solid lines in Figure 3.3) there is a second case: Intrusion type 21 was hypothesized which requires events of types 6 *again*, 17 *again*, and 23. For event types 17 and 23 there is no penalty because $(AE * I)_{17} \leq OV_{17}$ and $(AE * I)_{23} \leq OV_{23}$, and for event 6 there is no penalty either because the penalty was *already* taken into account for intrusion type 5. In this case, *where there should be a penalty, there is no penalty at all* (Diaz-Gomez and Hougen, 2005a). To address this problem Diaz-Gomez and Hougen (2005c) propose counting exactly the number of constraint violations made by each attack, which corresponds to the term T' in Figure 3.2.

Figure 3.2 gives, then, the AE matrix, an individual I hypothesized in the last generation, the counts of constraint violations for that individual T—as suggested by Mé (1998) for GASSATA—and the correct count of constraint violations T' as suggested by Diaz-Gomez and Hougen (2005b). However, it should be taken into account that T is raised to the power of 2 in Equation 3.1 and the parameter β is used to tune it; actually this is



Event	I	AE*I	OV	T
0	0	0	0	
1	0	0	0	
2	0	1	0	1
3	1	0	0	
4	0	0	0	
5	1	8	0	1
6	1	10	0	1
7	1	30	76	
8	1	5	0	1
9	0	0	0	
10	1	2	20	
11	1	3	0	1
12	1	0	0	
13	0	0	6	
14	0	0	0	
15	0	4	4	
16	0	0	0	
17	0	62	94	
18	1	100	0	1
19	0	5	42	
20	1	0	0	
21	1	0	0	
22	0	0	0	
23	1	5	5	
24		0	0	
25		3	459	
26		30	1335	
27		0	0	

Figure 3.3: Counting of overestimates for intrusions I_5 and I_{21}.

a problem in the sense that β has to be tuned for almost every file (i.e., observed vector) and with the side effect that if β is not well tuned, T^2 becomes quite high and then fitness values become negative unless α is incremented (if many fitness values are negative and proportional selection is used, then, many chromosomes are going to dissapear from the population).

Table 3.2 summarizes results with the same parameters and different data sets. The tendency is quite similar to that seen in Table 3.1: the average of false positives is large[3] and there is a false negative average overall (except when the input is an observed vector that does not have intrusions). An improved fitness function is needed in order to overcome these difficulties (see Section 3.3.2).

[3]The percentage for false positives is greater than 100% because 100% corresponds to the real number of intrusions, i.e, three, four, zero and twenty four, depending of the specific case.

	10 Run Average			Average %		
Input File	False +	False –	Detected	False +	False –	Detected
2051_7	8.9	0.2	2.8	297	7	93
2506_15	8.6	0.8	3.2	215	20	80
Zero Vector	10.0	0.0	0.0	Inf.	0	100
Full Vector	0.0	10.9	13.1	0	45	55

Table 3.2: Test with different input files; $\alpha = N_e^2/2$; $\beta = 1$; $500,000$ generations; 10 runs.

3.3.2 An Improved Fitness Function

As stated in Section 3.3.1 and Diaz-Gomez and Hougen (2005a), the term $\sum_{i=1}^{N_a} W_i * I_i$ was incorrectly guiding the fitness function, and the term T^2 was quite high when counting false hypotheses. As a first step towards an improved fitness function, consider the objective function:

1. Cut the positive side $\sum_{i=1}^{N_a} I_i$, and
2. Count overestimates in the correct way; this means, if two intrusions require excess occurrences of the same event, then count them twice, and so forth.

With this in mind, the fitness function only has one term, the penalty function,[4] and as the number of events is N_e, the new fitness function suggested is

$$F(I) = N_e - T'$$

. N_e corresponds to the total number of classified events. For testing, this value is 28. T' corresponds to the number of over counts for entry i, i.e., if $(AE \cdot I)_i > OV_i$.

It must be taken into account that the role of α corresponds now to N_e and that β is equal to one (see Equation 3.1). However, the term $\sum_{i=1}^{N_a} I_i$ was suppressed, as stated before. It must be reinforced that the reason for doing that is because the term $\sum_{i=1}^{N_a} I_i$ is giving the number of intrusions hypothesized but those intrusions have not been evaluated yet. In doing so, the evaluations may produce an incorrect count of overestimates, as can be seen in Figure 3.3.

Now, the hypothesized vector I is really evaluated in T'; the better the hypothesized vector, the smaller T' is, and of course, $F(I) \rightarrow N_e$, the maximum.

[4]In a similar work Li (2003) found a similar fitness function.

	10 run Average			Average %		
User	False +	False –	Detected	False +	False –	Detected
2051_7	0	0	3	0	0	100
2051_11	0	0	4	0	0	100
2506_15	0	0	4	0	0	100
Zero Vector	0	0	0	0	0	100
One Intrus.	0	0.1	0.9	0	0	90
Two Intrus.	0	0	2	0	0	100
Three Intrus.	0	0	3	0	0	100

Table 3.3: Robustness of the GA in doing misuse detection. Averaged over 10 runs.

In order to approximate a solution to the problem of finding the complete set of intrusions, as the GA runs it creates an aggregate solution set of all possible compatible types of abnormalities or intrusions found, using the *union operator*. The union operator functions as follows: (1) It records all the realistic solutions I (those where $T' = 0$) found in the search space while running. (2) Keeps track of each intrusion it finds within each realistic solution. (3) When a local maximum is found, the algorithm checks if the new intrusion already exists in its current solution set and if it does not, then it checks if it is mutually exclusive or not in order to add it to the corresponding aggregate solution set. In this way the algorithm builds up the set of all compatible abnormalities—the abnormal subset and the abnormal subset exclusive of the previous one.

The results found with the new fitness function and the union operator are shown in Table 3.3.

As can be seen, with the new fitness function *there are no false positives* and the number of *false negatives decreases dramatically*. This time 70 runs were performed, 10 for each type of data, and only one time a false negative was present.

This research has suggested some enhancements to the original work of Mé (1998) in order to avoid false negatives and false positives (Diaz-Gomez and Hougen, 2006a). For the false negative case, we have developed two mechanisms: the first one is the capturing of all possible intrusions in two subsets called the abnormal and abnormal exclusive subsets, which take into account the set of possible intrusions and the ones that in union with the set of possible intrusions violate the constraint. The second mechanism—quite related with the previous one—is the union operator, which stores and compares all the intrusions as the algorithm iterates. For the false positive case, we have suggested initially a fitness function that penalizes each chromosome taking into account intrusions that require the same event (see Section 3.3.1). Elaboration of these mechanisms and a dynamic fitness function is presented in Sections 5.1 and 5.1.1.

Chapter 4

State of Research on the Snake-in-the-Box Problem

The snake-in-the-box problem is a problem that can be stated in a simple way as: find the longest snake in a hypercube. At the same time, it is a problem that grows in complexity as the search space increases. The snake-in-the-box problem is so difficult that it is not known if the longest snakes found until now in hypercubes of dimensions $d > 7$ are actually the longest that can be found in those hypercubes. The snake-in-the-box problem, as a search problem of optimum paths, has been recognized as a NP problem due to the size of the search space and the constraints imposed (Bitterman, 2004). This dissertation has choosen this problem because it is a real problem of theoretical interest (Bitterman, 2004; Casella and Potter, 2004; Diaz-Gomez and Hougen, 2006b; Klee, 1970; Potter et al., 1994; Snevily, 1994) as well as practical in coding theory (Rajan and Shende, 1999), digital design, and telecommunications (Greenberg and Bhatt, 1990).

4.1 The Snake-in-the-Box Problem

A *d-hypercube* is a connected, non-directed graph of 2^d vertexes (or nodes), where each vertex has d neighbors and a binary labeling of each node may be given that differs in exactly one bit from that of each of its neighbors (Lakshmivarahan and Dhall, 1990). For example, Figure 4.1 shows a hypercube of dimension 4; it has 16 nodes, and each node is connected to 4 neighbors which differ in one base-two digit. For instance, node 0000 has 4 neighbors: 0001, 0010, 0100, and 1000.

A *snake* in a hypercube of dimension d is a *connected open path* in the hypercube where each node in the path has exactly 2 neighbors, except the *head* (source) and the *tail* (destination) that each have only one neighbor. The constraint is, then, that a node in the

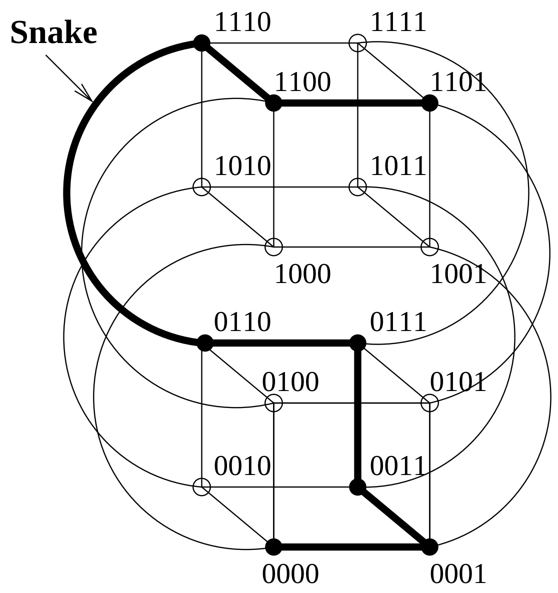


Figure 4.1: Hypercube of dimension 4 with a snake of length 7.

traversal is visited if it is not a neighbor of a previously visited node. Figure 4.1 shows a snake in a hypercube of dimension 4. The snake path is: $0000 - 0001 - 0011 - 0111 - 0110 - 1110 - 1100 - 1101$. Each node in the snake has exactly 2 neighbors, except nodes 0000 and 1101 that are the head and the tail. This snake has length 7 and is the longest in a hypercube of dimension 4 (Casella and Potter, 2004). Node 0100, for example, can not be in the snake because if it were in that path then it would have 3 neighbors: 0000, 0110, and 1100, violating the constraint[1] that the maximum number of neighbors must be 2, and there would be closed paths like $0100-0110-0111-0011-0001-0000-0100$, violating the constraint that the snake is an open path and that there should be two distinguished points each with only one neighbor in the path.

The length of a snake is the number of *edges* in the connected path.[2] Besides the complexity of the problem, one is faced with the fact that the lengths of longest snakes in hypercubes of dimension greater than 7 are unknown. Some theoretical upper bounds have been referenced (Weisstein, 2006) regarding *coils*, i.e., closed paths, or where the head is equal to the tail:

1. Klee (1970):

[1]Nodes 0110 and 1100 would violate this constraint as well, and node 0000 violates the heads' constraint.
[2]Some authors count the length as the number of vertexes or nodes.

$$\frac{7}{4(d-1)} \leq \frac{c(d)}{2^d} \leq \frac{1}{2} - \frac{1-12/2^d}{7d(d-1)^2+2} \text{ for } d \geq 6 \tag{4.1}$$

2. Abbott and Katchalski (1988):

$$c(d) \geq 77 \cdot 2^{d-8} \tag{4.2}$$

3. Snevily (1994):

$$c(d) \leq 2^{d-1}(1 - \frac{1}{20d-41}) \text{ for } d \leq 12 \tag{4.3}$$

and conjectured

$$c(d) \leq 3 \cdot 2^{d-3} + 2 \text{ for } d \geq 5 \tag{4.4}$$

where $c(d)$ corresponds to the length of the longest *coil*.[3]

To find the length of a snake $s(d)$, the difference between $c(d)$ and 2 is found; an approach that is used because of the lack of theoretical results with open paths.

The snake-in-the-box problem is a *search problem* in a search space of dimension 2^{2^d}, where d is the dimension of the hypercube. The search space then grows exponentially

[3]It should be noticed that Equation 4.4 was conjectured independently by Diaz-Gomez and Hougen (2006c) as a lower bound on the length of longest open paths in hypercubes of dimension d.

Hypercube Dimension	Snake Length
3	4
4	7
5	13
6	26
7	50
8	97
9	186
10	358
11	680
12	1260

Table 4.1: Lengths of longest known snakes. Casella and Potter (2004).

(2^{2^d}) as the dimension of the hypercube (d) increases. There have been some non-heuristic methods to solve the problem in hypercubes of dimension less than 8 (Casella and Potter, 2004) and at the same time, it has been shown experimentally that GAs are a powerful tool to solve this type of problem (Potter et al., 1994). Of course, experimentally researchers have been recording the longest snakes found, see Table 4.1 (Casella and Potter, 2004).

4.2 A Genetic Algorithm Approach to Hunting Snakes

With the explosion of the search space for the snake-in-the-box problem, a GA is an appealing tool to try to solve the problem approximately (Casella and Potter, 2004). As usual with GAs, besides the method to encode the possible solution, at least six parameters should taken into account: the population size, the fitness function, the selection mechanism, the crossover and mutation probability, and the stop criteria.[4] The parameters form a system; they are interrelated so a change in one could impact the quality of the solution or the number of computations required to obtain it (Diaz-Gomez and Hougen, 2007c).

For encoding the possible solution, basically one method has been applied (Bitterman, 2004): decimal encoding. With decimal encoding, two approaches have been used: (1) the labeling of the nodes from 0 to $2^d - 1$ (for example the snake in Figure 4.1, with this encoding, would be $0-1-3-7-6-14-12-13$); and (2) the labeling of the nodes using a Gray code (in which case the snake in Figure 4.1 would be $0-1-2-3-1-4-2-1$). With the first approach, a data structure that stores the connection of nodes is needed in order to guarantee a connected path; with the second one, a such data structure is not needed because the Gray code links adjacent nodes (Potter et al., 1994). However, the most common encoding used by GAs, binary encoding, has not been used, to our knowledge, except in our work (Diaz-Gomez and Hougen, 2006c,b), perhaps because the information regarding the connection of nodes is needed and/or the length of the chromosome grows exponentially (2^d).

The crossover and mutation probabilities used previously obey the standard for GAs: crossover in a range from 0.6 to 0.95, and mutation probability of 0.0, or in the range from 0.01 to 0.04 (Potter et al., 1994). However, there is no other justification for their use

[4]This dissertation is going to consider four in order to make the problem tractable.

Function	*Ave. Best Found*	*Maximum Length*	*Stdv.*
(1)	58.0	66	3.42
(2)	57.9	66	3.83
(3)	57.9	62	3.06

Table 4.2: Comparison of three fitness functions to hunt snakes reported by Bitterman (2004). 8-dimensional hypercube.

than empirical results. It should be highlighted that normally the operators used maintain the characteristics of snakes (Potter et al., 1994), so constructive factors are the only ones present.

As the snake-in-the-box problem is quite constrained, usually population sizes larger than $10,000$ individuals are used (Potter et al., 1994), with the common practice to embed the longest snakes found in a hypercube of dimension $d-1$ or the ones found in the d-dimensional hypercube. In order to evaluate the possible solutions a fitness function that takes into account the length of the snake is used (Potter et al., 1994), explicit formulation of fitness functions are reported as: (1) the length of the snake to the power of three, (2) the cube of the longest sub-snake plus the square of the length of smallest sub-snakes and (3) an evaluation of the length of the snake based on the narrowness of its path (Bitterman, 2004). Bitterman (2004) performed 40 runs for searching snakes in a 8-dimensional hypercube maintaining constants all parameters except the fitness function. Comparative results of these three fitness functions are shown in Table 4.2 taken from Bitterman (2004), where there is almost no difference in the average of longest snakes found.

According to Casella and Potter (2004), selection mechanisms like proportional selection, q-tournament selection, and rank selection have been tested, with results that show that rank selection has outperformed the other two (Casella and Potter, 2004), Bitterman (2004) reported over 40 runs results in Table 4.3, where in average, there is almost no difference on the selection mechanism used. Finally, as stop criteria, an approximate number of generations of $50,000$ or when there is no progress in the evolutionary process are used (Casella and Potter, 2004).

It should be emphasized that usually an hybrid approach has been taken in order to find snakes. Casella and Potter (2004) uses the population-based stochastic hill-climber *PBSHC*, where the snake is built gradually at each generation, there is no crossover, and the next node to growth is chosen at random from the possible nodes available in the

Selection	*Ave. Best Found*	*Maximum Length*	*Stdv.*
2-Tournament	58.9	63	2.56
3-Tournament	57.6	61	3.10
Roulette	58.1	66	3.44
Rank	58.4	66	3.88

Table 4.3: Comparison of selection mechanisms to hunt snakes reported by Bitterman (2004). 8-dimensional hypercube.

snakes' path. This means that at each generation all snakes have the same length. therefore, in order to perform selection the tightness[5] of the snake is used. Bitterman (2004) uses GA alone just for comparison purposes with hybrids like GA and depth first search and GA and narrowest path heuristic.[6] Bitterman (2004) states that GA alone are not good for finding snakes, but that combining GA with traditional search methods are more promising. However, as our research is in canonical GAs, we use only GAs not hybrids (see Section 6.3.3).

[5]Tightness is defined as the total number of valid nodes available in the hypercube to be used by the building snake (Casella and Potter, 2004).

[6]In narrowest path heuristic the algorithm performs n-look ahead in order to look for better paths (Bitterman, 2004).

Chapter 5

Theoretical Results

This dissertation addresses the difficult problem of setting certain parameters for GAs: the fitness function, the initial population, and the probabilities of crossover and mutation. This does not mean that other parameters, like selection pressure and stopping criteria are not important; but because each parameter itself constitutes an entire research topic, some have not been addressed.

The setting of parameters for GAs constitutes an optimization problem itself. Here, it raises the question: What is an optimal parameter setting (plan τ) for the solution of a problem that could guide the algorithm to get better solution solution quality? Parameter setting is usually approached by practitioners and researchers as providing empirical settings. Usually one looks for "good" solutions, but there may be a lack in the analysis of parameter setting.

The set of parameters for GAs constitutes a system. A change in one or more can impact others, but, besides that, it can impact the performance of the algorithm. Performance is measured in better solution quality and/or in the number of generations to reach a solution (Diaz-Gomez and Hougen, 2007c) (see Chapter 6 for empirical results).

We hope, then, to contribute to the research community with the present study, making a theoretical approximation for the setting of some parameters, the results of which are expected to be generalized.

5.0.1 Possible Factors that Influence the Initial Population

The first step in the functioning of a GA is, then, the generation of an initial population. Each member of this population encodes a possible solution to a problem. After creating

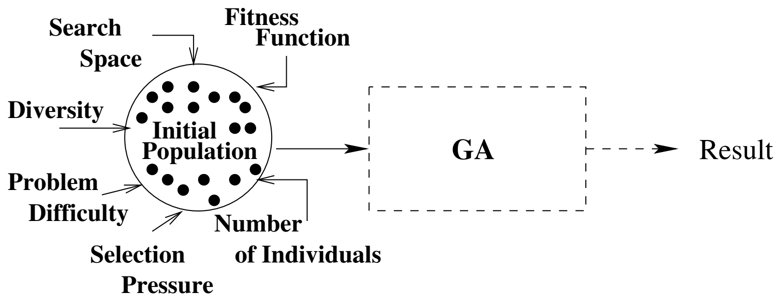


Figure 5.1: Some factors to take into account when the initial population is generated randomly.

the initial population, each individual is evaluated and assigned a fitness value according to the fitness function. However, this dissertation approaches the first research statement having in mind that the problem of finding a good initial population and the optimal population size is a hard problem (Eiben et al., 1999) and a general rule can not be applied to every type of problem or function to be evaluated (Lunacek and Whitley, 2006).

Figure 5.1 shows some factors that influence the initial population or that should be taken into account when an initial population is generated randomly: the search space, the fitness function, the diversity, the problem difficulty, the selection pressure, and the number of individuals.

The search space is influencing the initial population in the sense that usually there is a direct relation between search space and population size, the bigger the search space is, the more individuals the GA should have. However, if the problem is quite difficult and some information regarding the possible solution is available, then it is good to seed the GA with that information (Casella and Potter, 2004; Rajan and Shende, 1999), i.e., the initial population is seeded with some of those possible solutions or partial solutions of the problem. But, besides quantity, it has been recognized that if the initial population to the GA is good, then the algorithm has a better possibility of finding a good solution (Burke et al., 2004). A measure of diversity plays a role here in the sense that, when there is no information regarding a possible solution, then we could expect, that the more diverse the initial population is, the greater the possibility to find a solution is, and of course, the number of individuals in the population to get a good degree of diversity becomes important.

The selection pressure should be taken into account in the initial population size (Yu et al., 2006). One can say that, if a selection pressure Sp_1 is greater than a selection pressure Sp_2, then, when using selection pressure Sp_1 the population size should be larger than when using selection pressure Sp_2, because a higher selection pressure can cause a decrease in diversity (Harik and Lobo, 1999) of the population at a greater rate, perhaps causing the algorithm to converge prematurely.

The fitness function can be taken into account, in the sense that, the fitness evaluation of the initial population can be used as a metric of diversity (Burke et al., 2004; Diaz-Gomez and Hougen, 2007c), looking, for example, at the initial standard deviation of fitness values and evaluating the dispersion of such values or generating the initial population randomly and uniformly in the fitness value range (if bounded) (Diaz-Gomez and Hougen, 2007c).

Over previous stated factors, the one that is going to be studied in this dissertation is diversity. It is widely accepted that diversity plays an important role in the performance of GAs (Jaroslaw Arabas and Mulawka, 1995; Bitterman, 2004; Burke et al., 2004; Lobo and Lima, 2005; Grefenstette, 1986; McPhee and Hopper, 1999; Rosca, 1995). However, the research community has not formally defined diversity nor they have formally tested this hypothesis. There for, in this dissertation, we attempt to formalize various metrics to measure diversity and propose the first general hypothesis:

$$\text{If } V(P_A) \geq V(P_B) \text{ then } X(G, P_A) \geq X(G, P_B) \tag{5.1}$$

where $V(P_A)$ is the diversity of population P_A and $X(G, P_A)$ is the expected performance of a genetic algorithm G with population P_A. Expected performance is measured as the expected solution quality of the best solution found so far after a given number of generations or the expected number of function evaluations to obtain a solution of a given quality. Diversity is measured in terms defined in Section 5.0.2.

5.0.2 Metrics to Evaluate Diversity

This dissertation consider basically three types of measures for a fixed length population of chromosomes: one at the gene-level, one at the chromosome-level, and one at the population level. At the gene-level, diversity is measured at each locus of the entire population;

at the chromosome-level, diversity is measured in each chromosome of the entire population; and, at the population level, the position of each bit of each chromosome of the entire population is pondered.

5.0.2.1 Grefenstette Bias

In order to find *diversity* at the gene-level in a population, a formula that can be used is the *bias* measure suggested by Grefenstette (cited by Bäck (1996)) that is defined as

$$b(P(t)) = \frac{1}{l \cdot N} \sum_{j=1}^{l} max \left\{ \sum_{i=1}^{N} (1 - a_{i,j}), \sum_{i=1}^{N} a_{i,j} \right\} \tag{5.2}$$

where $b(P(t))$ is the $bias$ of the population $P(t)$ at time step t; l is the length of the chromosome, N is the number of individuals in the population, and $a_{i,j}$ is the j-gene of the i-individual.

Diversity as in Equation 5.2 is in the range $[0.5, 1.0]$; the nearer to 0.5 the more diverse the population is (Bäck, 1996). As it seams intuitive range diversity from $[0.0, 1.0]$, where values near to 1 are those that are more diverse, this dissertation suggest a slight change to Equation 5.2 to

$$d(P(t)) = 2 * (1 - b(P(t))) \tag{5.3}$$

Equation 5.2 is such that if N increases, then the term $max\{\sum_{i=1}^{N}(1-a_{i,j}), \sum_{i=1}^{N} a_{i,j}\}$ $\rightarrow N/2$ because in an initial random generation of genes it is expected to have half zeroes and half ones. Formally, if $N \rightarrow \infty$ and the initial random generation of genes is uniformly distributed, then

$$b(P(0)) = \frac{1}{l \cdot N} \sum_{j=1}^{l} max \left\{ \frac{N}{2}, \frac{N}{2} \right\} = \frac{1}{l \cdot N} \frac{N \cdot l}{2} = \frac{1}{2} \tag{5.4}$$

That corresponds to $d(P(0)) = 1$ in Equation 5.3, a fact that shows that in order to obtain "good" diversity—according with Grefenstette formula—the number of individuals in the population should be "big enough."

It should be reinforced that Equation 5.3 can not be seen as if all individuals in the population are different. If, for example, in an initial population of 8 individuals, 4 individuals are 11111111 and 4 individuals are 00000000, then $d(P(0)) = 1.0$ (the best), but it turns out that there are only two types of individuals. However, all positions of the chromosome have been represented with the possible values 0 and 1. If, on the contrary, all individuals in the initial population are different, as is the case of a base in an n-dimensional space, then $d(P(0)) = 2 * \{1 - [(n-1) \cdot l / l \cdot n]\} = 2 * \{1 - [(n-1)/n]\}$, clearly equal to 1.0 only when $n = 2$. These examples show some considerations that should be taken into account in Equation 5.3, if applied alone to measure diversity in the initial population (Diaz-Gomez and Hougen, 2007c).

5.0.2.2 Gene-Level Entropy

One common measure of uncertainty is *entropy*, defined as (Shannon, 1948)

$$H_j = -\sum_{i=1}^{N} p_{ij} * log_2 p_{ij} \tag{5.5}$$

where p_{ij} is the probability of occurrence of independent event $a_{i,j}$, and N is the number of trials (that is the population size in this case).

A measure of entropy at the gene-level for a population could be, then, the equation

$$H(P(0)) = \frac{1}{l} \sum_{j=1}^{l} H_j, \tag{5.6}$$

where H_j corresponds to the entropy as in Equation 5.5 for locus j of the entire population.

Looking at entropy values for different chromosome lengths, we find the same patterns as just discussed in Section 5.0.2.1—as l and N increase, entropy values are quite similar independently of l and N. However, both metrics, Equation 5.3 and Equation 5.6 scale different. Equation 5.5 (which is part of Equation 5.6)—takes into account the probability of occurrence of $1's$ and $0's$ for each locus in the entire population. On the contrary, Equation 5.3 takes into account the maximum number of $1's$ or $0's$, whichever it is, for each locus of the entire population. For example, for 3 bits with values 1, 0, 0, according to equation 5.5, $H = -(\frac{1}{3} log_2 \frac{1}{3} + \frac{2}{3} log_2 \frac{2}{3}) = -(-0.5283 - 0.39) = 0.9183$ and, according to Equation 5.3, $d = 2 * (1 - \frac{2}{3}) = \frac{2}{3} = 0.6667$ (Diaz-Gomez and Hougen, 2007c).

5.0.2.3 Chromosome-Level Hamming Distance

The *Hamming distance* measures the number of bits at which two individuals differ; it is defined as (Bäck, 1996; Frederick et al., 1993):

$$\rho_H(C_1, C_2) = \sum_{k=1}^{l}(|C_{1,k} - C_{2,k}|), \tag{5.7}$$

where C_1, C_2 are chromosomes and, $C_1, C_2 \in \{0, 1\}^l$.

In order to calculate an average Hamming distance, the Hamming distance between each pair of chromosomes in the population is needed. The Hamming distance of a population of size N, averaged by the total number of computations ($(N-1) * N/2$) is

$$\Gamma_H(P(0)) = \frac{2 * \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \rho_H(C_i, C_j)}{N * (N-1)} \tag{5.8}$$

For a uniformly randomly generated population, the average Hamming distance tends to $l/2$, because it is assumed that half of the bits are $1's$, half are $0's$, and that they are equally distributed. However, if in two chromosomes every bit is flipped, their Hamming distance is l. So the nearer to l the better. Cases where $\Gamma_H(P(0)) < l/2$ may be an indication of poor diversity at the chromosome level (Diaz-Gomez and Hougen, 2007c).

Taking again the example of 8 individuals, the first 4 being all genes $1's$ and the last 4 being $0's$, it is obtained $\Gamma_H(P(0)) = 2 * (8 * 4 + 8 * 4 + 8 * 4 + 8 * 4)/(8 * 7) = 4.5714$. This value is greater than $l/2$, and that could be considered as a good grade of diversity. However, if for the GA's purposes, there are only two types of individuals, having four of each type makes a difference.[1]

5.0.2.4 Chromosome-Level Neighborhood Metric

The Hamming distance with the concept of *neighborhood* defined by Bäck (1996) can be used, as a metric of diversity at the chromosome-level

$$B_k(C_m) = \{C_j \in \{0, 1\}^l | \rho_H(C_m, C_j) = k\} \tag{5.9}$$

[1]The difference is basically that the population size is different, and as such, it is expected that the performance of the GA is going to be different.

where $k \in \{0, 1, ..., l\}$ and $\rho_H(C_m, C_j)$ is as in Equation 5.7.

Equation 5.9 gives the neighbors of a chromosome C_m at a distance equal to k. In other words, if, for example, we want to find the individuals equal to C_m, then $k = 0$.

The chromosome-level neighborhood metric is defined as:

$$\mathcal{N}(C_m) = \frac{\sum_{k=0}^{l} |B_k(C_m)| * k}{\sum_{k=0}^{l} |B_k(C_m)|} \tag{5.10}$$

with C_m a pivot chosen from the population, $|B_k(C_m)|$ the cardinality of $B_k(C_m)$, i.e., the number of neighbors of C_m at a distance k in the whole population.

It is appreciated—using Equation 5.10—that as the population size grows, the tendency of the chromosome-level neighborhood is toward a distance of $l/2$, there is a concentration of points around the median point C_m at a distance $k = l/2$.

Again take the example of a population of 8 individuals, 4 of which are $1's$ and the last ones are $0's$. In order to calculate the neighborhood $B_k(C_m) = \{C_j \in \{0, 1\}^l | \rho_H(C_1, C_j) = k\}$, a pivot is needed; that could be any chromosome in the initial population. If the midpoint used is the chromosome of all $0's$, then C_m is going to have 3 neighbors at a distance $k = 0$, and 4 neighbors at a distance $k = 8$, then the average neighborhood distance as in Equation 5.10 is $(3 * 0 + 4 * 8)/(3 + 4) = 4.5714$; that, for this particular case, is exactly the same as the Hamming distance measure calculated in Section 5.0.2.3. However, the computational complexity of the chromosome-level neighborhood metric is $\mathcal{O}(N * l)$ against $\mathcal{O}(N^2 * l)$ of the Hamming distance metric.

5.0.2.5 Population-Level Center of Mass

Before a theoretical formulation of this metric, let us present an example of an initial population as in Table 5.1, where the center of mass $(\overline{x_1}, \overline{y_1})$ with respect to 1 is going to be calculated. For calculating the $\overline{x_1}$ coordinate, the column positions of gene 1 in row 1 is counted, that is $1 + 2 + 0 + 4$, plus the positions of gene 1 in row 2 that is $1 + 2 + 3 + 0$, plus the positions of gene 1 in row 3 that is $1 + 0 + 3 + 0$, and finally the positions of gene 1 in row 4 that is $0 + 0 + 0 + 0$ in order to obtain a total of 17, which is divided by the number of $1's$ in the population (8) to obtain $\overline{x_1} = 2.125$. Now, in order to calculate the $\overline{y_1}$ for gene 1, the row position of gene 1 in all columns is counted, which simplifying gives as a result: $1 + 2 + 3 + 0$, $1 + 2 + 0 + 0$, $0 + 2 + 3 + 0$ and $1 + 0 + 0 + 0$, for a total of 15. Again, the total is divided by the number of $1's$ in the population to obtain

1	1	0	1
1	1	1	0
1	0	1	0
0	0	0	0

Table 5.1: Initial population as a matrix of $1's$ and $0's$.

$\overline{y_1} = 15/8 = 1.875$). The center of mass with respect to gene 0 is calculated in a similar way, but taking into account gene 0 instead of gene 1.

Theoretically, population-level diversity metric looks at all genes in a population as a matrix of two types of genes (particles) and calculates the center of mass with respect to an origin $(0, 0)$—that is located at the left-top of the matrix. For the case of the x coordinate of the center of mass for a gene with value 1, it is suggested

$$\overline{x}_1 = \frac{\sum_{i=1}^{N} \sum_{j=1}^{l} T(a_{i,j})}{\sum_{i=1}^{N} \sum_{j=1}^{l} a_{i,j}} \tag{5.11}$$

where

$$T(a_{i,j}) = \begin{cases} j, \text{ if } a_{i,j} = 1 \\ 0, \text{ otherwise} \end{cases}$$

and $\sum_{i=1}^{N} \sum_{j=1}^{l} a_{i,j}$ is the number of those genes (i.e., where $a_{i,j} = 1$).

In order to obtain the y coordinate for a gene of type 1 a similar equation is suggested

$$\overline{y}_1 = \frac{\sum_{j=1}^{l} \sum_{i=1}^{N} R(a_{i,j})}{\sum_{j=1}^{l} \sum_{i=1}^{N} a_{i,j}} \tag{5.12}$$

where $R(a_{i,j})$ is the *row position* i of gene $a_{i,j}$ where the gene has value 1, and $\sum_{j=1}^{l}\sum_{i=1}^{N} a_{i,j}$ is the number of such genes as in Equation 5.11.

Equations 5.11 and 5.12 can be applied taking into account $0's$ instead of $1's$, in order to find the center of mass with respect to gene value 0. If the number of $1's$ and the number of $0's$ are uniformly distributed then it is expected that $(\overline{x_1}, \overline{y_1}) \approx (\overline{x_0}, \overline{y_0}) \approx (\frac{l}{2}+\frac{1}{2}, \frac{N}{2}+\frac{1}{2})$. The $\frac{1}{2}$ comes from the fact that the first gene is considered at a position $(1, 1)$.

Let us analyze the special case presented in section 5.0.2.1, where we have an initial population of 8 individuals, the first 4 of which are 11111111 and the last 4 individuals are 00000000. According to Equation 5.11—for the case of gene 1, $\overline{x}_1$ coordinate—$\sum_{j=1}^{8} T(a_{i,j}) = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36$ and, as there are 4 chromosomes with $1's$, then, the numerator is $36 * 4$. As $\sum_{i=1}^{N} \sum_{j=1}^{l} a_{i,j} = 8 * 4$ then, it is obtained $\overline{x}_1 = 36 * 4/8 * 4 = 4.5$. The same procedure applied in order to find the y_1 coordinate gives $\overline{y}_1 = 2.5$. This example gives as a result $(\overline{x}_1, \overline{y}_1) = (4.5, 2.5)$, i.e., the center of mass, for gene 1, is exactly in the middle coordinate of the first 4 chromosomes, with respect to origin $(0, 0)$. If the same calculation is performed for the center of mass for gene value 0, it is obtained $(\overline{x}_0, \overline{y}_0) = (4.5, 6.5)$. Clearly the $y's$ coordinates of the two centers of mass for 1 and 0, are showing an "unbalance" ($\overline{y}_1 = 2.5 \neq \overline{y}_0 = 6.5$) in the distribution of $1's$ and $0's$ in the matrix. At the x coordinate, the population is "balanced", i.e., $\overline{x}_1 = 4.5 = \overline{x}_0$.

This dissertation is suggesting to use this metric in *uniformly randomly generated genes* in a population because there could be cases—non randomly generated—where there is a perfect center of mass $(\frac{l}{2} + \frac{1}{2}, \frac{N}{2} + \frac{1}{2})$ for x and y coordinates but the population couldn't be diverse at all. As a matter of example, if we have $N = 11$ chromosomes with length $l = 11$, and genes with value 1 are in positions $(1, 1)$, $(1, 12)$, $(12, 1)$, and $(12, 12)$, and the rest of the genes are equal to 0, then the center of mass with respect to origin $(0, 0)$ is $(\overline{x}_1, \overline{y}_1) = (\overline{x}_0, \overline{y}_0) = (6, 6)$. However, extreme cases like a population equal to a base in the $\mathcal{B}^n$ space, is well captured by the center of mass, but not for other metrics just studied—see Section 5.0.2.1.

5.0.3 Analysis of Metrics

Studying metrics at gene level, similarities are observed among them. For instance, given any two individuals, both metrics give the same measure. However, when there are more than two individuals, the evaluation differs for all combinations where there are different proportion of $1's$ with respect to $0's$. Both present the same rank ordering but with different scale. The computation time needed in order to evaluate these metrics is $\mathcal{O}(N * l)$, where N is the size of the population and l is the chromosome length. These results suggest, that it does not matter which of these metrics is used.

Chromosome-Level neighborhood metric—as in Equation 5.10—takes less computation time ($\mathcal{O}(N * l)$) than Chromosome-level Hamming distance diversity $\mathcal{O}(N^2 * l)$, a fact that leads us to suggest using it at the Chromosome-Level. Both metrics are computing diversity at the chromosome level and both are using the Hamming distance; this makes the corresponding chromosome-level diversity values quite similar—at least for the current study.

Population-level diversity can capture the distribution of $1's$ and $0's$ in the whole population taking into account the structure of it. In order to see this, let us take the example of three individuals 101, 000, 101 (Deb and Jain, 2004). In this case the Grefenstette metric as in Equation 5.3 evaluates to 0.4444, the Entropy metric as in Equation 5.6 evaluates to 0.6122, the Hamming metric as in Equation 5.8 evaluates to 1.3333, the Neighborhood metric evaluates to 1.3333, and the center of mass with respect to $1's$ and $0's$ evaluates to $(2, 2)$. If the structure is changed as 110, 110, 000, then all metrics evaluate the same except the center of mass, that evaluates $(\overline{x}_1, \overline{y}_1) = (1.5, 1.5)$ and $(\overline{x}_0, \overline{y}_0) = (2.4, 2.4)$. The center of mass with respect to 1 and the center of mass with respect to 0 should tend to the middle of the matrix, and if that is not the case, that could be an indication that there is some bias toward a specific region of the search space, as this is the case.[2] However, what is a benefit for center of mass, could be a bias in small populations, as the case where we have a small change like 101, 010, 101 where the center of mass with respect to $1's$ and $0's$ is the same as in 101, 000, 101. The computational complexity of population diversity is $\mathcal{O}(N * l)$, where N is the size of the population and l is the chromosome length.[3]

5.0.4 Conclusions

The evaluation of diversity in a random population, using some of the metrics analyzed in this section, depends on the population size and chromosome length. However, as the population size and chromosome length increase, diversity becomes independent of them, and this is part of the difficulty of sizing an initial random population using only this approach. So, for small population sizes, one of the previous methods could be applied and, perhaps for big population sizes, one could measure diversity in a small portion of it or try to generate chromosomes uniformly distributed in the fitness landscape.

This chapter has suggested evaluating the initial population and, depending of the problem we are solving, we can choose gene-level diversity, chromosome-level diversity,

[2]Naturally, as the search progresses, the search is guided by the fitness function to find the correct solution, however, possibly an initial bias of the initial population could mislead the solution, or lower the search of the correct one.

[3]It is expected that if the computation time is lower for one metric with respect to other, the one with lower computation is preferred.

population-level diversity, or a combination of those, having in mind that some metrics are more computationally expensive than others. Some measures of diversity have been proposed, being maybe the prominent ones the neighborhood because it outperforms in computational complexity the Hamming distance metric, and population-level diversity that measures gene-level and chromosome level at the same time, with the $\overline{y}$ and $\overline{x}$ coordinate of the center of mass respectively, and that can be computed in $\mathcal{O}(N * l)$. It is expected that this theoretical approach is going to help us to address the first research statement regarding the influence of diversity in the initial population and the performance of some GAs.[4]

5.1 Parameter Tuning in Fitness Functions

A fitness function to be maximized is needed to construct a GA—the combination of objectives and constraints in a single function using arithmetic operators seems to be an appropriate way to define it. There are, however, problems with this approach. The first is that accurate scalar information must be provided on the range of objectives, to avoid some of them dominating the others (Section 3.3.1 presented a case study in this regard). The second is the difficulty in determining appropriate weights when there is not enough information about them. In this case, any optimal point obtained will be a function of the coefficients used to combine the objectives and constraints (Coello, 1998).

As an approximation to avoid the tuning of external parameters for some linear fitness functions (see Equation 5.15), we use internal parameters, i.e., parameters that belong to the problem itself, then, there is no external tuning required, just the fitness function that must be logically constructed (see Equation 5.14). The second hypothesis of this dissertation is then: If G_{F_1} is a GA with a fitness function with internal parameters which uses the union operator and G_{F_2} is a GA with a fitness function with external parameters, then

$$X(P_A, G_{F_1}) \geq X(P_A, G_{F_2}) \tag{5.13}$$

[4]Details of this research question will be given in Section7.1.

where $X(P_A, G_{F_1})$ is the expected performance of the GA with fitness function F_1 with internal parameters and with the union operator, and $X(P_A, G_{F_2})$ is the expected performance of the GA with fitness function F_2 with external parameters; expected performance is defined to be the expected solution quality after a given number of generations.

In order to perform specific test of the general hypothesis 2, the off-line intrusion detection problem is tested with F_1 Equation 5.14 and F_2 Equation 5.15. F_1 is used in conjunction with the union operator (see Section 3.3.2) in order to help the algorithm (G_{F_1}) to avoid false negatives.

$$F_1(I) = \frac{\sum_{j=1}^{N_a}(AE * I)_j - \sum_{j=1}^{N_a} max[0, (AE * I)_j - OV_j]}{\sum_{j=1}^{N_a}(AE * I)_j} \tag{5.14}$$

with N_a the number of attacks, AE the misuse matrix, I the chromosome, OV the observed activity and $\sum_{j=1}^{N_o}(A * I)_j \neq 0$ for $1 \leq j \leq N_o$ (Diaz-Gomez and Hougen, 2006a).

$$F_2(I) = \alpha + \sum_{i=1}^{N_a} W_i * I_i - \beta * T^2. \tag{5.15}$$

with $\alpha > 0$ and $\beta > 0$ external parameters to be tuned, N_a is the number of attacks and T the counting of fails as described in Section 3.3, and $W_i = 1$ for all $1 \leq i \leq N_a$ as described in Section 3.3.1.

Fitness function as in Equations 5.14 and 5.15 have been tested in order to find intrusions in audit trail files—see Section 6.2 and Mé (1998). Equation 5.14 has been tested too as part of some of some fitness functions proposed to find snakes in hypercubes—see Section 5.1.3.

5.1.1 A Dynamic Fitness Function

Equation 5.14 use only information that belongs to the solution of the problem itself, i.e., it has no external parameters and it can be rewritten in a more general form as

$$F(I) = 1 - \left(\frac{1}{\sum_{j=1}^{N_o} (A * x)_j} \right) \sum_{j=1}^{N_o} max[0, (A * x)_j - B_j] \quad (5.16)$$

and though of as $\alpha = 1$, $\beta = 1/\sum_{j=1}^{N_o}(A * x)_j$, where β is an *adaptive* parameter that depends on x, and $Penalty = \sum_{j=1}^{N_o} max[0, (A * x)_j - B_j]$ which takes into account the violation of the constraint in a *dynamic* way, i.e., it does not count the number of violations, but instead it finds, in a finer way, the positive difference of each $(A * x)_j$ from B_j and adds them up to get the net penalty for that particular x.[5] For example—see Section 3.3.1 and using AE as A, OV as B, I as x, N_e as N_o—if the target is the analysis of an observed vector that contains the count of a user's activity performed in a computer, then an individual of the population might incorrectly hypothesize with respect to one category of the observed vector. The number of faults, in this particular case, is one. However, how far (or near) was that fault? If the fault corresponds, for example, only to an entry in the observed vector which value is 0 and the individual is hypothesizing for that entry 299, then the distance is 299 because $\sum_{j=1}^{N_e} max[0, (AE * I)_j - OV_j] = 299$. On the contrary, if the hypothesis were a count of 1 (for the same specific entry), then the distance is 1 because $\sum_{j=1}^{N_e} max[0, (AE * I)_j - OV_j] = 1$. This is the type of dynamic differentiation the algorithm is performing. In the first case the individual is penalized with 299 and in the second case it is penalized with 1.

5.1.2 Additional Contributions

For the case of intrusion detection systems, we have worked to find a dynamic fitness function independent of external parameters that can be successfully applied given various data sets. Beyond that, we have continued with the union operator in order to reduce the number of false negatives and the number of generations to find optimums. So, having encountered a vector I of abnormality how can the algorithm come to a better solution that tries to get the maximum number of abnormalities or attacks? The next two sections deal with this topic and Sections 5.1.1 and 5.1.2.2 cover in more detail the benefits of Equation 5.14.

[5]Equation 7.3 was found independently by Fu and Wang (2005)—for a rule extraction system—in the general form $F(x) = 1 - E(x)$, where $E(x)$ is the classification error rate of chromosome x.

5.1.2.1 Abnormal and Abnormal Exclusive

In this study the set of abnormalities has been divided into two subsets: *abnormal* and *abnormal exclusive*. Abnormal exclusive is defined as those intrusions that can not occur at the same time as the abnormal subset. Depending on the Observed Vector in the analysis, if each type of attack is considered alone, it may satisfy $\sum_{j=1}^{N_e} max[0, (AE * I)_j - OV_j] = 0$, but if some attacks are considered together the constraint may be violated (i.e., $(AE * I)_j > OV_j$ for some j), then some of them are exclusive, i.e., can not occur at the same time. This is the case for intrusions that share some event type; for example, intrusions number 5, 19, and 21 share events number 6 and 17 (see Table 3.2); if the observed vector has, for instance, 8 events of type 6, then—looking only that entry[6]—intrusions 5 and 19 or 19 and 21 can occur—because $(AE * I)_6 = 6$ (which is less than or equal to $OV_6 = 8$) but not all together because in considering intrusions 5, 19, and 21, at the same time, $(AE * I)_6 = 11$ which violates the constraint because $(AE * I)_6 \leq OV_6$.

5.1.2.2 Pseudo Intrusions ς

An interesting topic to take into account in doing misuse detection is how near some activity is to abnormal (Diaz-Gomez and Hougen, 2006a). The fitness function proposed in Equation 5.14 gives the *similarity to intrusions*. For example, looking for individuals such that the constraint is not violated, (i.e., the fitness value $F(I)$ is equal to 1), then a set of currently possible intrusions can be obtained (intrusions that may have occurred during the period in question). However, if $F(I)$ is such that

$$\varsigma \leq F(I) < 1$$

with $\varsigma \gg 0$, then, there could be *pseudo intrusions* in the neighborhood ς. The set of pseudo intrusions is exclusive from the set of intrusions because the pseudo intrusion sets violate the constraint (see Section 5.1.2.1 and (Diaz-Gomez and Hougen, 2006a)).

[6]Other entries must be considered to get full intrusions 5, 19, and 21; for clarity entry 6 is the only one presented here.

5.1.3 Fitness Functions for Hunting Snakes

In order to try to generalize the application of the dynamic and generic Equation 5.14, this section is proposing some fitness functions to hunt snakes in hypercubes (see the definition of the problem in Section 4.1). The *objective* here is to find the longest snake, i.e., to maximize the number of nodes in the path. The *constraint* is to ensure that no characteristic of a snake is violated, i.e., to ensure that:

1. the path has exactly two end points—the head and the tail—with one neighbor each, and

2. the number of neighbors (VN_j) of each node j (other than the end points) belonging to the path is no greater than 2.

For the first constraint it should be noted that the expression "belonging to the path" means that the nodes (points) in the uni-dimensional array that represents the snake form a connected graph. There should not be *isolated points*, i.e., points marked as a 1 in the S array but unconnected (Diaz-Gomez and Hougen, 2006c). More generally, there should only be one connected path in the S array.

The second constraint takes into account whether there are no such distinguished points, or there is only one, or there are more than 2.

Longest snakes are those that have the maximum number of ones (nodes or points) in their chromosomes. Therefore, it is inferred that hypercubes containing longest snakes generally do not have many *lazy points*, i.e., points without neighbors (Diaz-Gomez and Hougen, 2006c). See Figure 5.2, where there is a chromosome 1101000000100100 with the path $0 - 1 - 3 - 11$, one isolated point (node 14), and one lazy point (node 13). However, if point 15 is added, the isolated point and the lazy point disappear and give rise to a snake $0 - 1 - 3 - 11 - 15 - 14$.

An appropriate fitness function that joins the objective (maximum length) and the constraints (retaining the properties of snakes) is needed.

5.1.3.1 A Normalized Fitness Function

As a first approach to joining the objective and the constraints, consider a normalized fitness function (Equation 5.17) that follows the guidelines of the fitness function in Equation 5.14 of Section 5.1:

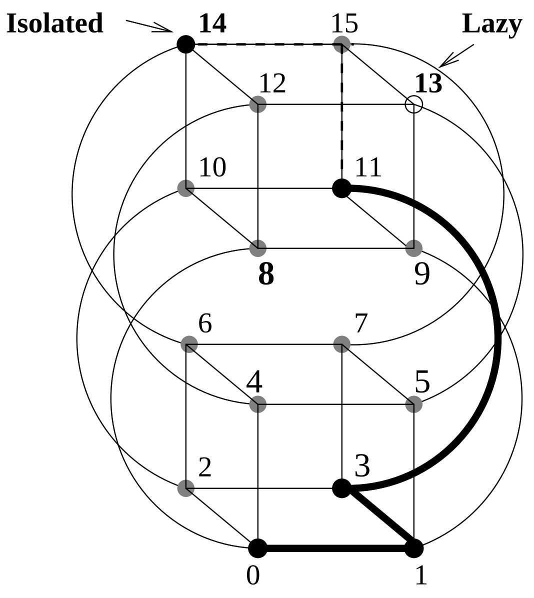


Figure 5.2: Isolated and lazy points in a 4-dimensional hypercube

$$F(I) = \left(\frac{\sum_{j=0}^{2^d-1} (AM * S)_j - Penalty}{\sum_{j=0}^{2^d-1} (AM * S)_j} \right) \left(\frac{Length(S) + 1}{\#P} \right). \tag{5.17}$$

where AM is the adjacency matrix that encodes the hypercube, S is the individual (solution) which is being evaluated, $Length(S)$ is the length of the longest snake in S,[7] $\#P$ is the number of points in the chromosome S, and

$$Penalty = \sum_{j=0}^{2^d-1} max[0, (AM * S)_j - 2] + \#Lazy + \#Isolated + |\#HeadTail - 2|$$

for j such that $S_j = 1$, i.e., where the first term $\sum_{j=0}^{2^d-1} max[0, (AM * S)_j - 2]$ takes into account those points j of the possible solution S that at the same time have gene value $S_j = 1$ and number of neighbors greater than 2; $\#Lazy$ is the number of lazy

[7]It should be emphasized that the term $Length(S)$ takes into account when a point in S—except the end points—has more than two neighbors. Besides that, if the path in S is unconnected, it results in a reduction of the length because just the longest snake in S is taken into account for calculating the length.

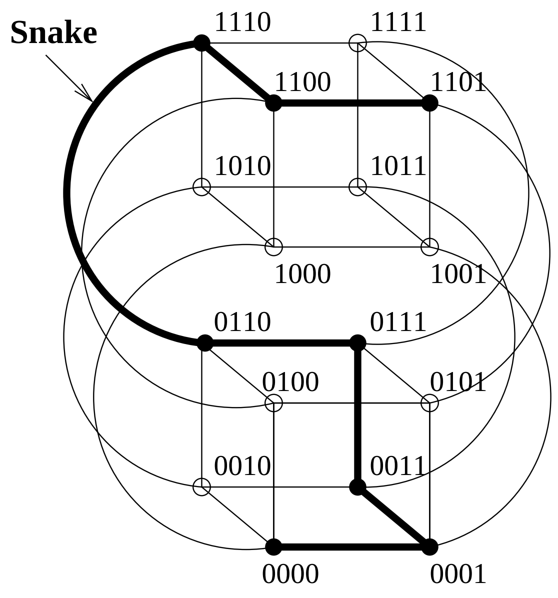


Figure 5.3: Hypercube of dimension 4 with a snake of length 7.

points; $\#Isolated$ is the number of isolated points; and $\#HeadTail$ is the number of distinguished points.[8]

Two terms should be emphasized in Equation 5.17:

- the first one—which is quite similar to Equation 5.14—takes into account the violation of the constraints, and
- the second $(Length(S) + 1)/\#P$, takes into account the connected path.

Whenever $F(I)$ of Equation 5.17 is equal to 1 chromosome S is a snake in the corresponding hypercube.[9] For example, in Figure 5.3 the snake $0000 - 0001 - 0011 - 0111 - 0110 - 1110 - 1100 - 1101$ in a 4-dimensional hypercube, satisfies

$$F(I) = \left(\frac{32 - 0}{32}\right)\left(\frac{7 + 1}{8}\right) = 1.$$

[8]In this problem there is not the longest snake to compare with and check for constraint violations—as there is for the misuse problem with the OV vector—so whenever S has a point j, with $S_j = 1$, in which $(AM * S)_j > 2$ then that counts for that difference in the first term of $Penalty$.

[9]There are many cases of hypercubes that do not contain snakes, examples like a chromosome S with no points, a single point, 3 isolated points, a closed loop, and so forth. Additionally if S contains a "snake" with an isolated point or additional unconnected paths, then S is not itself a snake.

Parameter	*Value*
Hypercube Dimension	4
Fitness Function	Normalized as in Equation 5.17
Max. generations	1,000
Type of Selection	Tournament Selection (75%-25%)
Initial Population	Randomly Generated
# Individuals	10
Chromosome Encoding	Binary
Chromosome Size	2^4
Crossover Probability	60%
Mutation Probability	3% per chromosome

Table 5.2: Plan τ for hunting snakes in a 4-Dimensional hypercube.

Each point of the possible solution S adds d to the sum in the first term in Equation 5.17 because each point is connected to d nodes in the d-hypercube. In this case, d is 4 and as there are 8 points belonging to the path, 32 is obtained as a result of $\sum_{j=0}^{15}(AM * S)_j$. $Penalty$ is equal to 0 in this case because there are no points in the snake with more than two neighbors, there are two distinguished points (0000 and 1101) with only one neighbor each, and there are no isolated or lazy points.

Another example is the path in Figure 5.2 where the fitness value is

$$F(I) = \left(\frac{20-1-1}{20}\right)\left(\frac{3+1}{5}\right) = 0.72$$

because there are 5 points in the chromosome, one isolated point, one lazy point and $Length(S)$ is 3.

Test were done on a 4-dimensional hypercube with the experimental settings as in Table 5.2. Results on 10 runs are given in Table 5.3, where *Run* is the number of the corresponding run, *Fitness* is the maximum value of the fitness in that run, *Length* is the maximum length of the snake found in the corresponding run, *#Isolated* is the number of isolated points found in that chromosome, *#Lazy* is the number of lazy points in that chromosome, *#Bad* is the number of bad points—which are those that have more than two neighbors—and *#Distinguish* are the number of distinguished points (which must be equal to 2 in a snake). Ten individuals are generated randomly;[10] in each run the same initial population was used.

[10]Only ten because the search space is 2^{2^4}, which is quite small.

	Equation 5.17					
Run	*Fitness*	*Length*	*#Isolated*	*#Lazy*	*#Bad*	*#Distinguish*
1	1.0	6	0	0	0	2
2	1.0	6	0	0	0	2
3	1.0	6	0	0	0	2
4	1.0	6	0	0	0	2
5	1.0	6	0	0	0	2
6	0.96	6	0	1	0	2
7	1.0	6	0	0	0	2
8	1.0	7	0	0	0	2
9	1.0	5	0	0	0	2
10	1.0	6	0	0	0	2
Totals						
10	9.96	60	0	1	0	20

Table 5.3: Results with fitness function as in Equation 5.17.

With the normalized fitness function as in Equation 5.17, on 10 runs, from Table 5.3 the following is observed:

- The fitness function reaches the maximum value (1) in 90% of the cases.
- In all cases the algorithm finds finds chromosomes S that are snakes, including the case where the fitness value is 0.96, because the lazy point does not violate the constraint, i.e., lazy points are not good for finding longer snakes, but they can occur.
- The algorithm finds the longest snake in run 8.
- The algorithm has no way to differentiate based on the lengths of the snakes, i.e., a snake of length 6 is as fit as a snake of length 7 because the fitness value is 1.0 in both cases.
- In all runs the algorithm finds the two distinguished points.

5.1.3.2 A Length-Differential Fitness Function

In order to see if a fitness function can usefully differentiate between snakes of different lengths, and to see if the number of longest snakes can be improved, Equation 5.17 was

	Equation 5.18					
Run	*Fitness*	*Length*	*#Isolated*	*#Lazy*	*#Bad*	*#Distinguish*
1	5.81	6	1	0	0	2
2	7.0	7	0	0	0	2
3	6.0	6	0	0	0	2
4	5.81	6	1	0	0	2
5	6.0	6	0	0	0	2
6	7.0	7	0	0	0	2
7	7.0	7	0	0	0	2
8	5.81	6	1	0	0	2
9	5.81	6	1	0	0	2
10	5.81	6	1	0	0	2
Totals						
10	62.05	63	5	0	0	20

Table 5.4: Results with fitness function as in Equation 5.18.

changed slightly. The second factor of Equation 5.17 is changed so that the length of the snake is not normalized by the number of points in the array S, giving

$$F(I) = \left(\frac{\sum_{j=0}^{2^d-1} (AM * S)_j - Penalty}{\sum_{j=0}^{2^d-1} (AM * S)_j} \right) * Length(S). \tag{5.18}$$

The same type of tests as in Subsection 5.1.3.1 was performed taking into account now that $0 \leq Length(S) \leq 7$, Table 5.4 shows the corresponding results and (Diaz-Gomez and Hougen, 2006b):

- The fitness function reaches maximum a fitness value (7) in 30% of the cases.
- In 50% of the cases the algorithm finds chromosomes S that are snakes; in the other 50% the graph of the chromosome is unconnected—to make S into a snake the isolated point must be removed.
- The algorithm finds a longest snake on 3 runs.
- The algorithm has a way to differentiate lengths of snakes.
- In all runs the algorithm finds the two distinguished points.

	Equation 5.19					
Run	*Fitness*	*Length*	*#Isolated*	*#Lazy*	*#Bad*	*#Distinguish*
1	7.0	7	0	0	0	2
2	7.0	7	0	0	0	2
3	7.0	7	0	0	0	2
4	6.0	6	0	1	0	2
5	6.0	6	0	1	0	2
6	6.0	6	0	1	0	2
7	6.0	6	0	1	0	2
8	7.0	7	0	0	0	2
9	7.0	7	0	0	0	2
10	6.0	6	1	0	0	2
Totals						
10	65.0	65	1	4	0	20

Table 5.5: Results with fitness function as in Equation 5.19.

5.1.3.3 A Single-Length-Dependent Fitness Function

The length of the snake is a good distinguishing factor for snakes, so one can use only that factor as a fitness function (Diaz-Gomez and Hougen, 2006b):

$$F(I) = Length(S). \tag{5.19}$$

One may think that perhaps no evaluation of the constraint is present in Equation 5.19 but this is not the case because when the length of the snake is calculated the algorithm begins in a distinguished point (head or tail) and it follows the path until the constraint is violated. The results obtained now are quite similar to the ones found in Section 5.1.3.2, but there are differences as well—see Table 5.5:

- The fitness function reaches a maximum (7) in 50% of the cases.
- In 90% of the cases the algorithm find snakes; in the other 10% the graph of the chromosome was unconnected.
- The algorithm finds a longest snake on 5 runs.
- The algorithm has a way to differentiate lengths of snakes.
- In all runs the algorithm found the two distinguished points.

5.1.3.4 A Quadratic Fitness Function

In Equation 5.19 lazy points are not penalized but there are some alternatives to do that; one is to choose a quadratic fitness function based on the number of points in the chromosome:

$$F(I) = (\#P - \#Lazy) * Length(S). \tag{5.20}$$

This equation is called quadratic because as $Length(S)$ is a function of the number of points, Equation 5.20 is quadratic in the number of points ($\#P$). Results are shown in Table 5.6.

This time some interesting things happen:

- The fitness function reaches a maximum (56) in 50% of the cases.
- In 50% of the cases the algorithm finds snakes; in the other 50% the graph of the chromosome is unconnected.
- When the algorithm finds snakes, those found are longest, i.e., the algorithm finds a longest snake on 5 runs.
- The algorithm has a way to differentiate lengths of snakes.
- In 90% of the runs the algorithm finds the two distinguished points.

This brief description shows how a parameter can mislead the algorithm.

5.1.3.5 A Linear Fitness Function that Takes into Account Lazy Points

Starting from the fitness function in Equation 5.19 it is possible to penalize the lazy points in a different way than what is done in Equation 5.20 (see Section 5.1.3.4):

$$F(I) = Length(S) - \#Lazy. \tag{5.21}$$

Now the results are shown in Table 5.7 including:

- The fitness function reaches a maximum (7) in 70% of the cases.

	Equation 5.20					
Run	*Fitness*	*Length*	*#Isolated*	*#Lazy*	*#Bad*	*#Distinguish*
1	48	6	1	0	0	2
2	56	7	0	0	0	2
3	56	7	0	0	0	2
4	48	6	1	0	0	2
5	13	1	0	0	9	1
6	56	7	0	0	0	2
7	56	7	0	0	0	2
8	48	6	1	0	0	2
9	56	7	0	0	0	2
10	48	6	1	0	0	2
Totals						
10	485	60	4	0	9	19

Table 5.6: Results with fitness function as in Equation 5.20.

	Equation 5.21					
Run	*Fitness*	*Length*	*#Isolated*	*#Lazy*	*#Bad*	*#Distinguish*
1	6	6	0	0	0	2
2	7	7	0	0	0	2
3	6	6	1	0	0	2
4	7	7	0	0	0	2
5	7	7	0	0	0	2
6	7	7	0	0	0	2
7	6	6	0	0	0	2
8	7	7	0	0	0	2
9	7	7	0	0	0	2
10	7	7	0	0	0	2
Totals						
10	67	67	1	0	0	20

Table 5.7: Results with fitness function as in Equation 5.21.

- In 90% of the cases the algorithm finds snakes; in the other 10% the graph of the chromosome is unconnected.
- The algorithm finds a longest snake on 7 runs.
- The algorithm has a way to differentiate lengths of snakes.
- In all runs the algorithm finds the two distinguished points.

Success at finding longer snakes improves by 20% compared with results as in Equation 5.19 and all chromosomes show snakes except one—see run 3 where there is an isolated point.

5.1.3.6 A Linear Fitness Function that Takes into Account Lazy and Isolated Points

As good results were obtained with the linear function in Equation 5.21, a new linear function that takes into account both lazy and isolated points is going to be proposed:

$$F(I) = \mathit{Length}(S) - \#\mathit{Lazy} - \#\mathit{Isolated} \tag{5.22}$$

Results are shown in Table 5.8, including:

- The fitness function reaches a maximum (7) in 60% of the cases.
- In 90% of the cases the algorithm finds snakes; in the other 10% the graph of the chromosome is unconnected.
- The algorithm finds a longest snake on 6 runs.
- The algorithm has a way to differentiate lengths of snakes.
- In all runs the algorithm finds the two distinguished points.

Contrary to what might be expected, there are not better results than with Equation 5.21 because this time 6 longest snakes are obtained vs. 7 as is shown in Tables 5.7 and 5.8. The new term $\#Isolated$ in Equation 5.22 did not fix it at all, as in shown in entry 7 of Table 5.8.

	Equation 5.22					
Run	*Fitness*	*Length*	*#Isolated*	*#Lazy*	*#Bad*	*#Distinguish*
1	6	6	0	0	0	2
2	7	7	0	0	0	2
3	6	6	0	1	0	2
4	7	7	0	0	0	2
5	7	7	0	0	0	2
6	7	7	0	0	0	2
7	6	6	1	0	0	2
8	6	6	0	0	0	2
9	7	7	0	0	0	2
10	7	7	0	0	0	2
Totals						
10	66	66	1	1	0	20

Table 5.8: Results with fitness function as in Equation 5.22.

5.1.3.7 A Rational Fitness Function

As the objective of the snake-in-the-box problem is to find *longest snakes* two factors should be considered: (1) *longest*, which means with a maximum number of points, and (2) *snakes*, which is the constraint—see Section 5.1.3. This leads to the idea of a *rational fitness function*, i.e., a fraction where the numerator is the objective and the denominator is the constraint. In this way if the numerator increases, then the fraction increases, and if the denominator decreases, then the fraction increases, accomplishing the maximization of the fitness function. The fitness function proposed is, then,

$$F(I) = \frac{Length(S)}{1 + Penalty} \tag{5.23}$$

where $Penalty$ is as defined in Section 5.1.3.1.

Results are shown in Table 5.9, including:

- The fitness function reaches a maximum (7) in 30% of the cases.
- In all cases the algorithm finds snakes.
- The algorithm finds a longest snake on 3 runs.
- The algorithm has a way to differentiate lengths of snakes.
- In all runs the algorithm finds the two distinguished points.

Run	*Fitness*	*Length*	*#Isolated*	*#Lazy*	*#Bad*	*#Distinguish*
1	7	7	0	0	0	2
2	6	6	0	0	0	2
3	6	6	0	0	0	2
4	7	7	0	0	0	2
5	3	6	0	1	0	2
6	6	6	0	0	0	2
7	3	6	0	1	0	2
8	3	6	0	1	0	2
9	3	6	0	1	0	2
10	7	7	0	0	0	2
Totals						
10	51	63	0	4	0	20

Table 5.9: Results with fitness function as in Equation 5.23.

Comparing with Equation 5.17 which gave as result all snakes, as it is the case, fitness function as in Equation 5.23 obtained longest snakes in 30% of the tests against 10% compared to Equation 5.17 in Section 5.1.3.1 and the average length was 6.3 against 6.0 as in Equation 5.17—but this time there are 4 lazy points in the 10 runs against 1 as in Equation 5.17 (See Table 5.9). Linear Equations as in 5.19, 5.21 and 5.22 outperformed Equation 5.23 in finding longest snakes and in average—see Tables 5.5, 5.7 and 5.8.

5.1.4 Additional Contributions

One issue regarding the snake-in-the-box problem is that the length of longest snakes in hypercubes of dimension greater than 6 is unknown. Some theorist have suggested and justified mathematical bounds for such length for the case of coils (see Section 4.1 and we have conjectured a lower and an upper bound for the case of snakes (Diaz-Gomez and Hougen, 2006b). This topic, the way the length of the snake can be evaluated and some open questions are going to be outlined in this section.

5.1.4.1 Mathematical Conjectures on the Snake-In-The-Box Problem

We define the *energy* of a snake (coil) as the dot product between the proposed snake (the chromosome S) and the product vector *VN* which is the result of the matrix multiplication between the adjacency matrix *AM* (see Section 6.2.3) and the chromosome S. Formally,

d	*Energy*	*#P Conjecture 1*	*Abbott*	*Casella*
3	8	-	2.4	5
4	14	8	4.8	8
5	26	14	9.6	14
6	50	26	19.2	26
7	98	50	38.5	50
8	194	98	77.0	97
9	386	194	154.0	186
10	770	386	308.0	358
11	1,538	770	616.0	680
12	3,074	1,538	$1,232.0$	1,260

Table 5.10: Conjecture 1 of lower bound in the number of points in longest snakes vs. theoretical (Weisstein, 2006) and empirical (Casella and Potter, 2004) findings.

$$E = S \cdot (AM * S)^T = S \cdot (VN)^T. \tag{5.24}$$

The energy of a snake (coil) is independent of d (Diaz-Gomez and Hougen, 2006c).

For instance, for snake $\{0, 1, 3, 7, 6\}$ the energy is the scalar value $E = (1\,1\,0\,1\,0\,0\,1\,1) \cdot (1\,2\,3\,2\,2\,2\,1\,2)^T = 8$.

Conjecture 1. The number of points in the longest snake in a hypercube of dimension d where $d > 3$, is greater than or equal to the energy of the longest snake in the hypercube of dimension $d - 1$ (Diaz-Gomez and Hougen, 2006c).

Table 5.10 shows the values of the energy for known longest snakes and the corresponding number of points for each snake. We use the energy definition *inductively*, i.e., beginning with an already known number of points of a longest snake in the 3-dimensional hypercube, the number of points of a longest snake in the 4-dimensional hypercube can be conjectured, and so forth. Table 5.10 compares conjecture 1 with theoretical results from Abbot (Weisstein, 2006) and empirical results (Casella and Potter, 2004). Values quite near between conjecture 1 and both the experimental findings and the theoretical bounds are observed.

Conjecture 2. A *lower bound* for the number of points of the longest snake in a hypercube of dimension d, with $d > 3$ is

$$3 \cdot 2^{d-3} + 2.$$

This conjecture holds directly from the energy of a snake.[11]

Conjecture 3. An *upper bound* for the number of points of the longest snake in a hypercube of dimension d is (Diaz-Gomez and Hougen, 2007b)

$$\#P \leq 2^{d-1} - \sum_{i=1}^{d-3} i + 1, \quad \text{for } d \leq 7 \tag{5.25}$$

and

$$\#P \leq 2^{d-1} - \sum_{i=1}^{d-2} i + 1, \quad \text{for } d > 7 \tag{5.26}$$

This conjecture is related to the conjecture of the number of points that a longest snake can traverse in the next dimensional hypercube. For example, if a $(d-1)_l$-dimensional hypercube[12] has a longest snake of length n, then the maximum number of points that can be used in the next $(d-1)_u$-hypercube is $(2^{d-1} - n)$, because 2^{d-1} is the number of new nodes given by the $(d-1)_u$-hypercube and n cannot be used because those were already used in the $(d-1)_l$-hypercube. The question is if n nodes of the $(d-1)_u$-hypercube has been invalidated by the longest snake in the $(d-1)_l$-hypercube, how many new nodes are going to be invalidated in the new $(d-1)_u$-hypercube as the snake is growing? Suppose those are going to be m nodes, the number of points of the snake in the d-hypercube would be $\#P = (n+1) + 2^{d-1} - n - m = 2^{d-1} - m + 1$. Where m is the number of new nodes that are invalidated by the snake in the $(d-1)_u$-hypercube and that corresponds to the $\sum i$ term in Equations 5.25 and 5.26. In general, if a point p is chosen in the snake path, that point has d neighbors, one of which can be chosen as the next link in the path—if p is not the last one and if the next point $p+1$ does not violate the constraint—then, p can invalidate $(d-1)$ nodes if it is a head or a tail or at most $(d-2)$ otherwise. Table 5.11 compares theoretical results (Weisstein, 2006) with coils[13] and practical founding as in (Casella and Potter, 2004) for dimension $d \leq 12$.

[11]This result was independently conjectured by Snevily (1994) as an *upper bound* for the length of longest *coil* of dimension $d \geq 5$. We conjectured this as a *lower bound* for *snakes*.

[12]A d-dimensional hypercube can be seen as two $d-1$-dimensional sub-hypercubes: $(d-1)_l$-hypercube and $(d-1)_u$-hypercube

[13]For comparison purposes and because of the lacking of theoretical findings for bounds for snakes (open paths).

D	*Conjecture*	*Klee*	*Snevily*	*Casella*
3	5	-	-	5
4	8	-	-	8
5	14	-	14	14
6	27	32.0	26	27
7	55	63.9	50	51
8	101	127.9	98	98
9	221	255.9	194	187
10	468	511.8	386	359
11	970	$1,023.7$	770	681
12	$1,983$	$2,047.6$	$1,540$	1261

Table 5.11: Upper bound of number of points of longest snakes in hypercubes. Klee (1970) and Snevily (1994) correspond to upper bounds for coils. Casella and Potter (2004) corresponds to empirical findings for snakes.

5.1.4.2 Snakes' Fitness Evaluation

The way the length of a snake is evaluated can impact the performance of the algorithm in terms of the number of generations required to find an optimum. For example, if the algorithm is evaluating the length of the chromosome $01010001101000110110111010010001$ in a 5-dimensional hypercube (see Figure 5.4), then it could start counting the length by using one of the three points: 18, 24, or 27. If the algorithm just finds one starting point (head or tail) and begins to count until the constraint is violated, then, if the algorithm begins with point 18, then the length is going to be 7 which corresponds to the path $18-22-20-21-17-1-3-7$ because point 15 has three neighbors in the connected path, i.e., it is violating the constraint. If the algorithm begins to count the length starting at point 24 then the length is going to be 3 which corresponds to the path $24-8-10-14$. Finally, if the algorithm begins to count the length starting at point 27, the length is going to be 1 because the connected path without violation of the constraint is going to be $27-31$. So, the actual chromosome could be evaluated with different lengths, depending on the starting point the fitness function is using in the evaluation[14] impacting the performance of the algorithm in the number of generations to find snakes, as is shown in Section 6.2.4.1.

[14]Another possibility of evaluation is to take the sum of all lengths. However, there is not always a point like 15 in the example that makes the path connected.

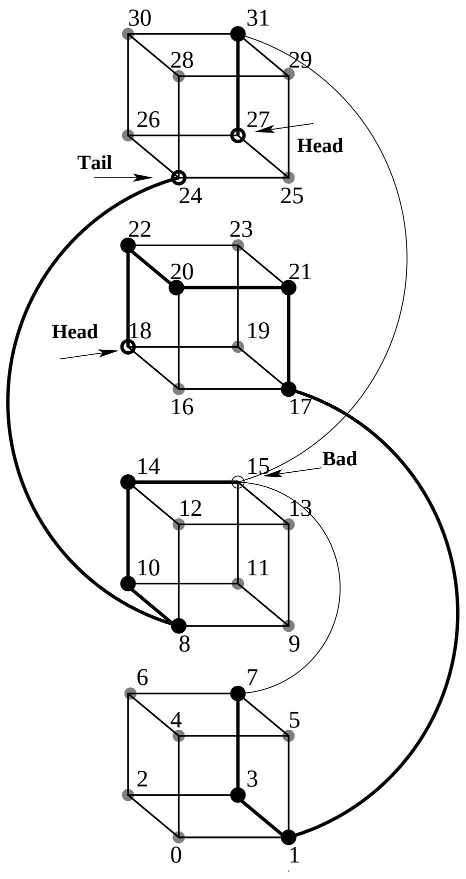


Figure 5.4: Fitness evaluation of the length of a snake.

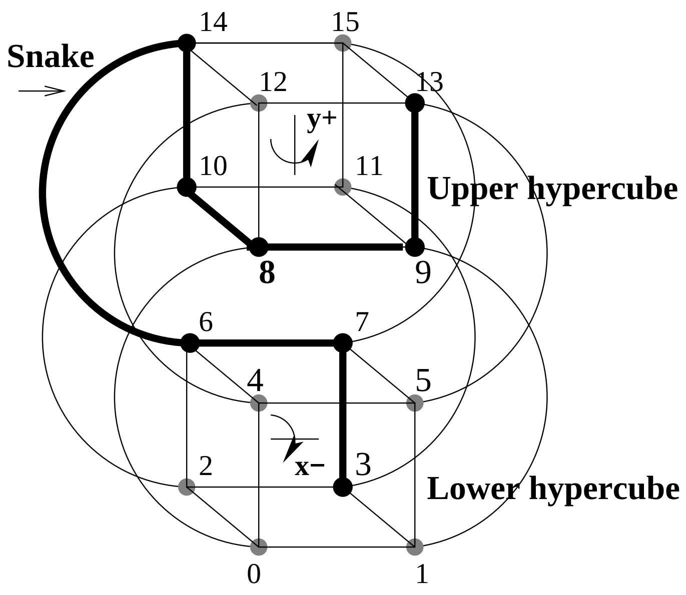


Figure 5.5: Snake in a 4-dimensional hypercube to be converted to a coil.

5.1.4.3 Open Questions

Several open questions remain (Diaz-Gomez and Hougen, 2006b). One question is, given the length of a snake in a d-dimensional hypercube, how many different snakes does the hypercube contain of that length, excluding possible isomorphisms? The reason for this question is that, depending on the number of snakes (solutions), finding one can be more or less difficult.[15]

A second question can be formulated as, is the longest coil (a closed path that otherwise conforms to the constraints of a snake) minus one point the longest snake in the d-dimensional hypercube?[16]

The next interesting topic related to snakes and coils is how, by doing rotations and/or translations, a snake can be converted to a coil. For example, see Figure 5.5 which is a snake. The lower 3-dimensional hypercube is rotated $-90°$ around the x axis, and the upper 3-dimensional hypercube is rotated $+90°$ around the y axis to obtain the coil in Figure 5.6. It should be taken into account that the rotations and/or translations are done but the numbering remains stationary. For instance, in the previous example the coil is $7 - 5 - 4 - 12 - 8 - 9 - 11 - 15 - 7$, as is shown in Figure 5.6.

[15]One possible answer could be 0 snakes of that specific length.

[16]Coils, like snakes, are also of practical interest.

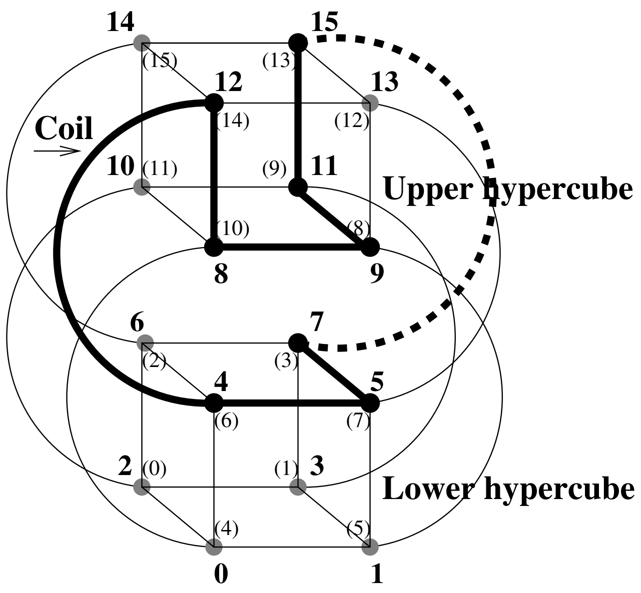


Figure 5.6: Snake in Figure 5.5 converted to a coil by two rotations.

5.1.5 Conclusions

A GA needs a fitness function that combines objectives and constraints into a single value (Coello, 1998). The problem is not only to find the appropriate form for the function, but also to provide accurate values for the parameters so that they will produce the correct solution in as many instances of the problem as possible. Objectives and constraints are given in Equation 5.14. $\sum_{j=1}^{N_o}(A * x)_j$ is the *objective*—the more intrusions the better. $\sum_{j=1}^{N_o}\max[0,(A * x)_j - B_j]$ is the *penalty* if the *constraint* is violated. However, as the function in Equation 5.14 is normalized, it is useful for finding local minima or maxima (Diaz-Gomez and Hougen, 2006a). Knowing this fact, we propose the use of the union operator, so that each time the algorithm finds a local maximum, it is stored—avoiding false negatives—and it is tested in conjunction with previous maxima obtained (see Section 3.3.2). We have performed an empirical study for the case of the intrusion detection problem in order to test the validity of this assumption (see Sections 6.2 and 3.3.2). We found found a speed up well in excess of an order of magnitude (Diaz-Gomez and Hougen, 2006a).

It is highlighted that Equation 5.14 has no external parameters to be tuned, contrary to what is suggested in previous research (Mé, 1993, 1998). The function 5.14 is using information from the problem itself and penalizing each individual according to how far (or near) it is from the observed vector B (see Section 5.1.1).

Equation 5.14 was rewritten as Equation 7.3 (see Section 5.1.1), a form that indicates explicitly no violation of the constraint when $F(I) = 1.0$. This form (Equation 7.3) has been used by ourselves (Diaz-Gomez and Hougen, 2005c) and other researchers, not only in the same context of intrusion detection (Crosbie and Spafford, 1995; Li, 2003), but also in a different context (Fu and Wang, 2005). However, we emphasized again, not only the granularity of the penalty term in Equation 7.3, but also that it has been mathematically justified (see Section 5.1).

Equation 5.14 is used as part of certain fitness functions proposed for hunting snakes (see Section 5.1.3) in an effort to apply it in a different domain. However, in the snake-in-the-box problem, the *length* is of crucial importance because the GA is looking for the longest snakes. Therefore, the term *length(S)* is going to be in all the equations proposed (see Section 5.1.3). Again, all fitness functions proposed for trying to solve this problem use information from the problem itself; no external parameters have to be tuned. This problem is quite constrained: one head, one tail, longest connected path with exactly two neighbors in the path (except the head and the tail). If for example, one is using Equation 5.17 and a result is obtained as a path with 4 heads/tails, one could say that the term $|\#HeadTail - 2|$ of the penalty should have a greater weight. What would be a "good" value for that weight? Tests should be conducted in order to find such a value, and here is where the parameter setting comes into account using information from the problem itself in order to avoid external setting.

5.2 Crossover Rate vs. Mutation Rate

When a problem is encoded using GAs, one must address the number of possible structures $\mathcal{A}$ to be tested, the probability of crossover and mutation, the stopping criteria, the type of selection operator, and the fitness function to be used in order to solve the problem. This section focuses on the GA crossover and mutation operators, using as a tool the schema theorem (see Section 2.1), having in mind that the choice of $\tau's$ parameters and operators are problem dependent.

Looking at GA operators, it can be seen that one purpose of the crossover operator, for example, is to search prominent regions of the search space in order to climb to a maximum according with the fitness function given. Researchers usually use a probability of $60\% - 70\%$ for performing crossover (Mitchell, 1998), but why? How does the probability of crossover influence the quality of the possible solution or the number of iterations? With regard to the mutation operator, it gives the possibility to move a chromosome to a new region of the search space, it helps with diversity and in moving the algorithm from stationary points. That is, if we have a specific gene which has a value is of 1 for all individuals in a population, then in performing crossover that specific gene is never going to change. Mutation, then is the tool that can probabilistically change the value of that gene—in this case to 0. How could one figure out the probability value to be used for mutation? Is there a relation between the crossover ratio and the mutation ratio for some GAs? This Section is then answering the third research question regarding the relation between crossover and mutation probability.

As stated before, the approach taken for this study of the crossover and mutation rates is the schema theorem, which has been generalized by Goldberg and Sastry (2001) as the equation

$$E(\xi, t+1) \geq (1 - p_c \epsilon) S_p n(\xi, t) \tag{5.27}$$

where S_p is the selection pressure which plays the role of $\hat{\mu}_\xi(t)/\overline{\mu}(t)$ and $\epsilon = d(\xi)/(l-1)$ corresponds to the disruption of schema ξ.

As $n(\xi, t)$ is expected to be greater than 1, then, in order to obtain a non decreasing number of schema, Equation 5.27 should satisfy $(1 - p_c S_p \epsilon) \geq 1$ (Goldberg and Sastry, 2001), which gives an upper bound on the probability of crossover with respect to selection pressure of $p_c \leq \left(1 - S_p^{-1}\right)/\epsilon$ (Goldberg and Sastry, 2001). For example, if $\epsilon = 1$ and $S_p = 2$ then $p_c \leq 0.5$. This approach is used in this section, including for the mutation operator.

5.2.1 The Schema Theorem Trade-Off

The schema theorem, without considering the possibility p_ξ of crossing with the same schema, can be written as (Mitchell, 1998):

$$E(\xi, t+1) \geq \left(1 - p_c \frac{d(\xi)}{l-1}\right)(1-p_m)^h \, \frac{\hat{\mu}_\xi(t)}{\overline{\mu}(t)} n(\xi, t), \tag{5.28}$$

or as $E(\xi, t+1) \geq (1-p_c\epsilon)(1-p_m h)S_p n(\xi, t)$ when p_m is close to zero because if $p_m \approx 0$ then $(1-p_m)^h \approx (1-p_m h)$. This approximation allow us to see similarities between p_c as related to ϵ, as well as, similarities between p_m as related to h (see Equations 5.30 and 5.31). Schema ξ is expected to grow if

$$(1-p_c\epsilon)\,(1-p_m h)S_p \geq 1. \tag{5.29}$$

with $p_c\epsilon \neq 1$ and/or $p_m h \neq 1$ as is done by Goldberg and Sastry (2001) without mutation.

Of the three factors in Equation 5.29, the only one that could be greater than 1 is S_p, so selection pressure is the hand that guides the artificial evolutionary process. But, if the other two factors $(1-p_c\epsilon)$ and $(1-p_m h)$ can not be greater than 1, then both could be approximately 1 to try to increase the instances of ξ at $t+1$. To do that, $p_m h \approx 0$ and $p_c\epsilon \approx 0$. For $p_m h \approx 0$, as h, the number of defining bits, is usually ≥ 1, then p_m should be, in general, ≈ 0. For $p_c\epsilon \approx 0$, if $\epsilon \approx 1$, then $p_c \approx 0$, and if $\epsilon \approx 0$, then $p_c \approx 1$. Theoretically it could be concluded that besides S_p, in order to help schema ξ grow, if $h \geq 1$ then p_m should be maintained ≈ 0 and, if the schema is long ($\epsilon \approx 1$) then p_c could be ≈ 0, and if the schema is short ($\epsilon \approx 0$) then p_c could be ≈ 1 (Diaz-Gomez and Hougen, 2007f).

Formally, from Equation 5.29:

$$p_c \leq \frac{1}{\epsilon}\left(1 - \frac{1}{(1-p_m h)S_p}\right), \tag{5.30}$$

which constitutes the third general hypothesis of this dissertation: there is a relationship between selection pressure, crossover and mutation rates $< S_p, p_c, p_m >$ expressed in Equation 5.30, with $p_m h \neq 1$, $p_m \approx 0$, $h > 1$, and $S_p > 1$, where ϵ is the crossover disruptive factor, h is the number of defining bits, and S_p is the selection pressure of the corresponding plan τ that influences the evolutionary process.

As $1/(l-1) \leq \epsilon \leq 1$, then from Equation 5.30 we obtain:

$$p_c \leq \frac{1}{\epsilon}\left(1 - \frac{1}{(1-p_m h)S_p}\right) \leq (l-1)\left(1 - \frac{1}{(1-p_m h)S_p}\right)$$

.

Looking at this bound for short schema, and considering $(l-1) > 1$, then, in order to have p_c bounded[17] the term $\left(1 - \frac{1}{(1-p_m h)S_p}\right) \rightarrow 0$, in which case $p_c \approx 0$ and $p_m \approx (S_p - 1)/S_p h$, or $\left(1 - \frac{1}{(1-p_m h)S_p}\right) \rightarrow 1/(l-1)$, in which case $p_c \approx 1$, and if $l \gg S_p$ then $p_m \approx (S_p - 1)/S_p h$. This result shows the strong dependence of p_m on the selection pressure S_p and the number of defining bits h.

Considering long schema, i.e., where $\frac{d(\xi)}{l-1} = 1 = \epsilon$, then $p_c \leq 1 - 1/(1-p_m h)S_p$. In the limit, if $p_m \rightarrow (S_p - 1)/s_p h$ then $p_c \approx 0$, and if $p_m \rightarrow 0$ then $p_c \approx 1 - 1/S_p$.

The previous cases showed a limit approach maintaining $0 \leq p_c \leq 1$. A practical approach could be if $p_m = 0.001$, $S_p = 2$, $h = 2$, and schema is long, then $p_c \leq 0.4989$ (*low*).[18] Figure 5.7 shows a case of long schema with p_m constant and equal to 0.001. As the number of defining bits grow, p_c decreases; but, if the selection pressure increases, then p_c grows. For $S_p = 2$, $h = 512$ and $p_m = 0.001$, $p_c = -0.025$, this could be an indication that the probability p_m should be changed; for example if p_m is changed to 0.0009 then $p_c = 0.073$, maintaining $S_p = 2$ and $h = 2$.

An equation similar to Equation 5.30 can be obtained for the case of mutation:

$$p_m \leq \frac{1}{h}\left(1 - \frac{1}{(1-p_c\epsilon)S_p}\right), \tag{5.31}$$

that can be used as an upper bound on p_m.

If $h \geq 1$, then from Equation 5.31

$$p_m \leq \frac{1}{h}\left(1 - \frac{1}{(1-p_c\epsilon)S_p}\right) \leq \left(1 - \frac{1}{(1-p_c\epsilon)S_p}\right)$$

[17]Attention should be taken with this theoretical approach. As p_c and p_m are probabilities, then the corresponding range is $[0, 1]$. But, equations like 5.30 and 5.31 can give values > 1.0 or < 0.0 depending on the parameters used, values that could be interpreted as a possible change in one or more of the parameters involved or that could be constrained to specific bounds as suggested in this section.

[18]This corroborates that $p_c \rightarrow 1 - 1/S_p$ when $p_m \approx 0$.

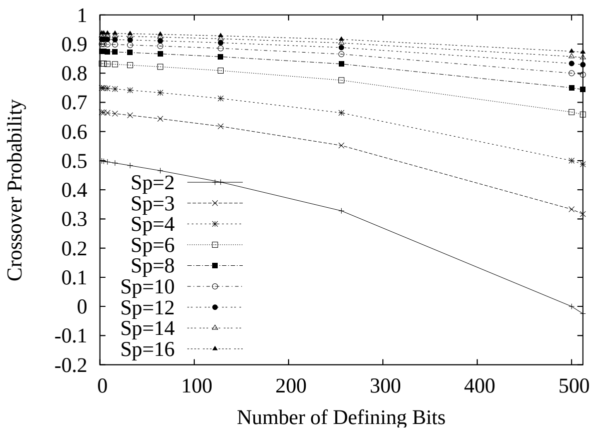


Figure 5.7: p_c when $p_m = 0.001$. Long schema.

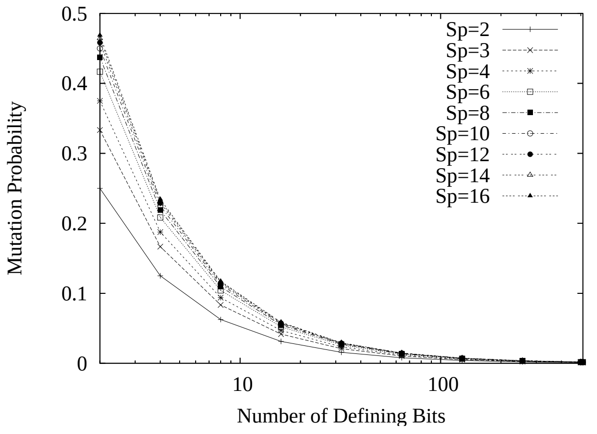


Figure 5.8: p_m when $p_c = 0.0$. Long or short schema. Log x option used.

is obtained. Looking at this bound, if $p_c \to 0$ then $p_m \to 1 - 1/S_p$, and if $p_c \to 1$ and ϵ is small, then $p_m \to 1 - 1/S_p$, which means that the bigger the selection pressure S_p is, the bigger the probability of mutation p_m should be.[19] Now, if h is considered in the analysis, then according to Equation 5.31, if $p_c \to 0$ then $p_m \leq (1/h)(1 - 1/S_p)$ which, besides considering the selection pressure S_p, shows that the more defining bits h there are, the lower the probability of mutation should be.

The previous cases showed a limit approach. A practical approach could be if $p_c = 0$, $h = 2$, and $S_p = 2$, then $p_m \leq 0.25$ independent of ϵ; but if $p_c \to 1$ then only small schema

[19]The case $p_c \to 1$ and ϵ long is not considered because $p_c\epsilon$ must be $\neq 1$ according to the initial premise stated in Equation 5.29.

can be considered, i.e., if $\epsilon = 1/(l-1)$, $h = 2$, and $S_p = 2$ then $p_m \leq (l-3)/(2(2l-4))$, so if l is large enough then $p_m \leq 0.25$. This shows an upper bound of 0.25 on mutation probability when $S_p = 2$ and $h = 2$. A similar approach can show different bounds depending on S_p, h, and p_c. Figure 5.8 shows a case with p_c constant ($= 0$). As the number of defining bits increases, p_m decreases, but if the selection pressure increases, then p_m increases (Diaz-Gomez and Hougen, 2007f).

5.2.2 Conclusions

This section suggest a guide to finding possible values for the probabilities of crossover and mutation for GAs, taking as a reference the schema theorem in order to answer the third research question. A trade-off of the schema theorem has been presented in which it can be seen that the parameters $\epsilon = d(\xi)/(l-1)$ and h (the number of defining bits) can be related to the difficulty of the problem. If $\epsilon \approx 1$ and $h \gg 1$, then the problem could be cataloged as a *difficult problem* and then p_c and p_m should be low. If $\epsilon \approx 1/(l-1)$ then the problem can be cataloged as moderate and then p_c could be high. If the number of defining bits is high (i.e., $h \gg 1$) maybe the more difficult the problem is and the mutation operator can be destructive, so the assumption is to maintain p_m small, depending on the selection pressure used.

For the present analysis, selection pressure is the hand that guides evolution: if there is not enough selection pressure, then, it is possible that the algorithm is not going to obtain "good" quality on the solution, but if the selection is quite increased without considering changes in crossover and mutation probabilities—i.e., if the schema theorem trade-off is not obeyed—then, it is possible that the quality of the solution is diminished, as is shown in some of the empirical studies presented in Section 6.3.4. As stated in Section 5 the setting of parameters constitutes an optimization problem itself. There are no fixed rules and, besides the crossover and mutation rates, others parameters can impact the quality on the solution as well. Empirical studies is conducted in Section 6.3 in order to see how the trade-off of the schema theorem presented in this section works.

Chapter 6

Exploratory Study

Trying to answer the three general questions outlined in Section 1.1, some empirical studies regarding (1) population size, (2) the use of internal parameters by a fitness function in an off-line intrusion detection system, and (3) the schema theorem trade-off were performed before the specific hypotheses were tested in order to be more confident with some of the problems and parameters used and that are stated in Chapter 7.

6.1 Initial Population

Many efforts have been made toward solving the problem of population size but, because population size depends in part on the difficulty of the problem to be solved (Harik and Lobo, 1999; Pelikan et al., 2000; Yu et al., 2006), it remains an open problem (Piszcz and Soule, 2006a). However, as population size is quite important for the efficiency of evolutionary algorithms (Jaroslaw Arabas and Mulawka, 1995; Costa et al., 1999; Piszcz and Soule, 2006a), various empirical methods have been proposed and some success reported (Eiben et al., 1999), like the use of self adaptation (Bäck et al., 2000; Harik and Lobo, 1999), which basically uses varying population size and may be the most prominent result until now. This dissertation presented in Section 5.0.1 a theoretical approach to population size using metrics at the gene-level, chromosome-level, and population-level, which are going to be used here to measure populations of some cases that have been reported as good populations, but also, to address the problem using some classical functions used in the GAs' literature, like the one-max and deceptive functions.

#	*Code/Chromosome*
1	1 0 1 1 1 0 0 0 1 0 0
2	0 1 0 1 1 1 0 0 0 1 0
3	0 0 1 0 1 1 1 0 0 0 1
4	1 0 0 1 0 1 1 1 0 0 0
5	0 1 0 0 1 0 1 1 1 0 0
6	0 0 1 0 0 1 0 1 1 1 0
7	0 0 0 1 0 0 1 0 1 1 1
8	1 0 0 0 1 0 0 1 0 1 1
9	1 1 0 0 0 1 0 0 1 0 1
10	1 1 1 0 0 0 1 0 0 1 0
11	0 1 1 1 0 0 0 1 0 0 1

Table 6.1: Case I: Structured Initial Population. Perfect Population Diversity, Center of Mass $(\mathbf{\overline{x}_1}, \mathbf{\overline{y}_1}) = (\mathbf{\overline{x}_0}, \mathbf{\overline{y}_0}) = (\mathbf{6}, \mathbf{6})$.

6.1.1 Error-Detecting Codes

Reeves (1993) presents an interesting case of a structured initial population for error detecting. Here the population size is small and the initial random generation of the population is not sufficient for the solution of the problem. The populations in Tables 6.1 and 6.2 show two cases (Reeves, 1993).

In Table 6.1 each column and each row has 5 ones and 6 zeros. Diversity measured at the gene-level is 0.9940. At the chromosome-level, the diversity is 6. Finally, at the population-level, $(\overline{x}_1, \overline{y}_1) = (\overline{x}_0, \overline{y}_0) = (6, 6)$. Looking at the gene-level, the measure is above 0.99; with a population size of 10, that can be considered good. This fact is reinforced with the values obtained at the chromosome-level and population-level being "perfect" (see Section 5.0.2). The pivot used in Equation 5.10 was the fifth member of the population in Table 6.1; it turns out that any pivot used has 10 neighbors at a distance of 6.

For comparison, 11 codes of length 11 were generated randomly 30 times. Table 6.3 shows the corresponding diversity values. None score as high in diversity as the ones obtained with the structured population. For example, $\overline{x}_0$ was always equal to 6.00 but that never was the case for $\overline{x}_1$;[1]. Wether this difference is practical significant remains to be seen.

[1]Not all $\overline{x}_1$ are in Table 6.3 however, none of them is equal to $\overline{x}_0$.

#	*Code/Chromosome*
1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
3	0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
4	0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
5	0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
6	0 1 1 0 0 1 1 1 1 0 0 1 1 0 0
7	0 1 1 1 1 0 0 0 0 1 1 1 1 0 0
8	0 1 1 1 1 0 0 1 1 0 0 0 0 1 1
9	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
10	1 0 1 0 1 0 1 1 0 1 0 1 0 1 0
11	1 0 1 1 0 1 0 0 1 0 1 1 0 1 0
12	1 0 1 1 0 1 0 1 0 1 0 0 1 0 1
13	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
14	1 1 0 0 1 1 0 1 0 0 1 1 0 0 1
15	1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
16	1 1 0 1 0 0 1 1 0 0 1 0 1 1 0

Table 6.2: Case II: Structured Initial Population. Perfect Gene Diversity, $\mathbf{H(P(0)) = 1.0}$.

Metric	*Maximum*	*Minimum*	*Average*	*Std. Dev.*
Entropy	0.98	0.83	0.93	0.03
Neighbors	5.44	4.60	5.33	0.15
(x_1, y_1)	(6.61,6.83)	(5.43,5.39)	(6.07,5.96)	(0.28,0.35)
(x_0, y_0)	(6.00,6.62)	(6.00,5.38)	(6.00,6.06)	(0.00,0.32)

Table 6.3: Measuring a Random Initial Population of $\mathbf{11}$ Individuals. Chromosome Length $\mathbf{11}$. $\mathbf{30}$ Runs.

In Table 6.2, the population has 8 ones and 8 zeros in each column, and 8 ones and 7 zeros in each row except for row 1 which has all zeros. Diversity measures at the gene-level result in a perfect value of 1.00; at the chromosome-level the diversity value is 8 (it should be noted that all members of the population are at a Hamming distance of 8, independent of the pivot used); and, at the population-level $(\overline{x}_1, \overline{y}_1) = (8, 9)$ and $(\overline{x}_0, \overline{y}_0) = (8, 8)$.

For comparison, 16 individuals of length 15 were generated randomly 30 times. The corresponding diversity values are in Table 6.4. There were not "good" diversity values like the ones obtained with the structured population in Table 6.2. For instance, the maximum gene-level diversity was 0.99 vs. 1.00 for the structured.

Metric	*Maximum*	*Minimum*	*Average*	*Std. Dev.*
Entropy	0.99	0.91	0.95	0.02
Neighbors	6.35	5.73	6.07	0.20
(x_1, y_1)	(8.47,9.38)	(7.38,8.10)	(7.99,8.59)	(0.31,0.31)
(x_0, y_0)	(8.00,8.92)	(8.00,7.67)	(8.00,8.42)	(0.00,0.30)

Table 6.4: Measuring a Random Initial Population of 16 Individuals. Chromosome Length 15. 30 Runs.

6.1.2 The One-Max Function

The one-max problem has been wide used in the lecture as a classical theoretical problem to be studied using GAs (Cheng and Kosorukoff, 2004; Harik and Lobo, 1999; Giguere and Goldberg, 1998; Yu et al., 2003). The problem is to find a chromosome of all ones starting from an initial population randomly generated. This problem has the characteristic that the schema under consideration is one: every single bit is going to contribute to the solution. In this problem there is only one global maximum and no local minima.

Harik and Lobo (1999) tested the one-max function using an optimal GA in order to compare results with a "parameter-less genetic algorithm." The optimal GA has the following parameters with no mutation involved: chromosome length 100, tournament size 2, and probability of crossover 1 (Harik and Lobo, 1999). These authors reported, on average, $2,500$ function evaluations with the optimal GA and $7,400$ with the "parameter-less genetic algorithm," performing 20 runs.

Our test is performed over 30 runs, i.e., using 30 different seeds. The tests run the algorithm until it reaches the maximum or gets stuck, i.e., a column of zeros is encountered. Figure 6.1 shows how the entropy measure—changing the population size—influences the average quality of the solution. For population size less than 42 (entropy values less than 0.98299) almost no run finds the optimum. For population sizes between 44 and 110 (entropy values between 0.983887 and 0.993427), at least one run finds the optimum. For population sizes between 112 to 144 (entropy values between 0.993573 and 0.995), almost all find the optimum. Finally, the algorithm always converges to the maximum for entropy values averaging above 0.995 for the test set described previously. But how the result turns out if the only variable that changes is diversity? This is the topic of hypothesis 1 as is going to be stated in Chapter 7. More exploratory study is in Diaz-Gomez and Hougen (2007a)

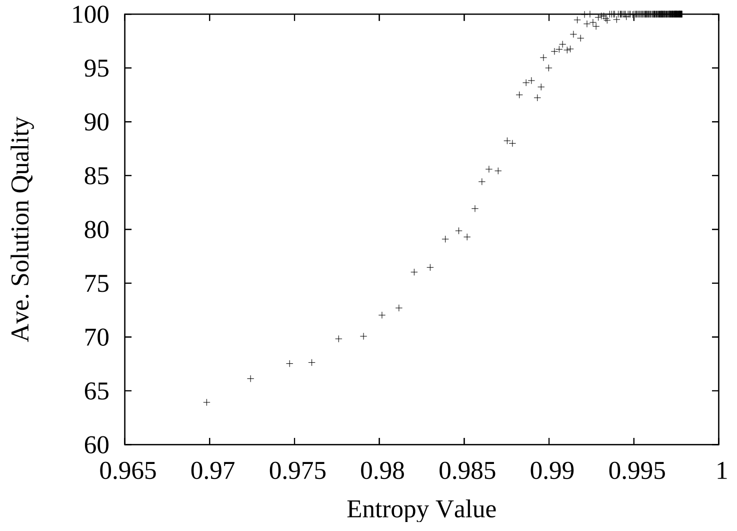


Figure 6.1: Entropy Measure vs. Quality of the Solution. One-Max Function. Population size changing. 30 Runs.

6.1.3 Conclusions

Some case studies were presented regarding the influence of diversity in the performance of GAs. The case of the error-coding examples (Reeves, 1993), could be seen in two ways. As a counter example to random initial populations, in the sense that the error-coding population was structurally generated, and as a perfect example of diversity because the error-coding has "optimum" diversity values according to some of the metrics presented in Section 5.0.2. It seems that for small populations it may be better to generate structured chromosomes than random ones (Reeves, 1993), and diversity can help to measure how structured the initial population is.

For the cases of the one-max function and deceptive functions (Diaz-Gomez and Hougen, 2007a), it is highlighted that a minimum initial population is needed in order to converge—in our experimental set up we obtain less than 42 individuals, i.e, entropy values less than 0.98 for the one-max function, and less than 22 individuals , i.e., entropy values less than 0.96—however, there is approximately an optimum size in the neighborhood of 100 for the one-max function (Harik and Lobo, 1999) and 64 for the deceptive one, in the sense that, usually, fewer individuals or more individuals than that neighborhood can cost more function evaluations to reach an optimum (global for the case of the one-max and local for the case of the deceptive function) or the divergence of the algorithm.

Deceptive functions and its variants (Diaz-Gomez and Hougen, 2007a) show a drawback in the random generation of the initial population, in the sense that if almost all chromosomes are going to be in the same neighborhood, i.e., where a local maximum is located, the GA is going to be trapped in it. This fact enforces the hypothesis that if there is no "good" diversity in the search space, it is possible for the algorithm to diverge or find a poor solution (see first research question in Section 1.1). However, this does not imply, in general, that a higher diversity automatically gives us a better performance in GAs. This is a natural consequence of the "optimal neighborhood" of population size where fewer or more individuals than the "optimal" can cause the algorithm to diverge, expend more computations, or find poor solutions. It is possible that not even with $1,000,000$ individuals in the population will the GA reach the global optimum because at that population size the maximum number of ones obtained in the chromosomes are less than or equal to 75, exactly where the inflexion or discontinuity occurs (Diaz-Gomez and Hougen, 2007a). Neither the crossover and mutation operators, nor the selection pressure, could produce the necessary changes to reach the global optimum when the population was created using random generation of genes. In order to get out of the trap, the initial random generation had to be changed taking into account a uniform distribution of fitness values in conjunction with a higher selection pressure.

However, it should be taken into account that the setting of the rest of the GAs' parameters influence the performance of the algorithm as well.

6.2 Parameter Tuning in Fitness Functions

This section is going to outline the benefit of using a fitness function with internal parameters as in Equation 5.14 and the union operator. A comparison between the fitness function proposed by Mé (1993), and the fitness function proposed by Diaz-Gomez and Hougen (2005c), and Diaz-Gomez and Hougen (2006a) is made in order to show the drawbacks of tuning external parameters. The three fitness functions are presented here again:

$$F(I) = \alpha + \sum_{i=1}^{N_a} W_i * I_i - \beta * T^2. \tag{6.1}$$

$$F(I) = N_e - T' \tag{6.2}$$

$$F(I) = \frac{\sum_{j=1}^{N_e}(AE*I)_j - \sum_{j=1}^{N_e} max[0,(AE*I)_j - OV_j]}{\sum_{j=1}^{N_e}(AE*I)_j} \tag{6.3}$$

The GA parameters that are going to be used are the ones used by Mé (1993): population size 50, crossover probability 0.60, mutation probability 0.0083 per bit, I length 24, and $1,000$ generations. Besides that, Mé (1993) utilizes a threshold D in order to differentiate an intrusion or a non-intrusion. For example, in the final population each locus of the entire population is analyzed: if the number of $1's$ is greater than D, then there is a possible intrusion. This is different from what is proposed in this dissertation in the sense that the algorithm is storing the best solution so far and, if the union operator is used, then when a new intrusion is found, it is checked to see if it is already in the current solution. If not, then it is checked to see if it violates the constraint in order to add the possible intrusion to the current solution subset or to add it to the exclusive abnormal subset (see Sections 5.1.2.1 and 3.3.2).

Equation 6.1 was tested with various $\alpha's$ and $\beta's$ (external tuning, Mé (1993) suggests $\alpha = 50.0$ and $\beta = 1.0$) and Equations 6.2 and 6.3, which use information from the problem itself, where tested and compared with Equation 6.1 as is shown in Table 6.5. As β is the factor that is influencing the penalty term (see Equation 6.1), if it is quite high (β_3) then the number of false negatives is high (7 with $\alpha_3 = 50$ and $\beta_3 = 7$). However, as β begins to decrease (β_3, β_4, and $\beta 5$ maintaining $\alpha = 50$ constant), the number of false negatives decreases, but the number of false positives begins to increase $3 - 6 - 32$ for previous $\beta's$. If α and β are decreased ($\alpha_3 = 4$ and $\beta_3 = 0.05$), then the number of false positives is huge (296). If α is increased as $\alpha_1 = N_e^2/2$ and $\beta_1 = 1.0$, the algorithm performed the best for external tuning in the present test set. How is this drawback conciliated? Equations 6.2 and 6.3 do not need tuning because they do not have external parameters;[2] however, the appearance of false negatives needs to be approached as is shown in the next section.

[2]For the case of Equation 6.2, one can think that N_e is external, however, that parameter is the number of events of the observed vector, i.e., the parameter is using information from the problem. If the number of events change, then N_e changes accordingly.

	Parameters			
Equation	α	β	*False +*	*False -*
6.1	392.0	1.0	5	0
6.1	4.0	0.05	296	1
6.1	50.0	7.0	3	7
6.1	50.0	1.0	6	0
6.1	50.0	0.5	32	0
6.2	(●)	-	0	39
6.3	(●)	-	0	28
6.2	($\star$)	-	0	0
6.3	($\star$)	-	0	0

Table 6.5: Difficulty in tuning of external parameters. Comparison of three fitness functions. (●): Without the union operator. ($\star$): With the union operator. 30 runs per parameter setting.

6.2.1 Avoiding False Negatives

The fitness function as is Equation 6.3 tries to avoid false positives. The GA proposed was tested with different inputs (different users and artificial intrusion sets). No false positives nor false negatives where found in these test sets when the union operator is used;[3] with the aggregate value that the union operator makes the algorithm converge quickly. Figure 6.2 shows that in testing done, on average $2,524.83$ generations were needed using the union operator, against $79,431.90$ without it. The test was done this time over 280 runs, and using different lengths for the intrusion vector (I) in order to get a general conclusion. The I vector used had lengths 24, 32, 40, and 48 (Diaz-Gomez and Hougen, 2006a); data are from the Lincoln Laboratory (Fried and Zissman, 1998) and in addition, artificial intrusion vectors are used to simulate more attacks.

In order to check the computability of the algorithm, the length of the hypothesized vector (individual) is crucial for the applicability of the algorithm and fitness function proposed. If vector I has length 24, the possible number of solutions is 2^{24}, which is of a magnitude of 16 million. If it has a length of 32, the possible number of solutions is 2^{32}, which is a magnitude of billions; if it has a length of 40 the possible number of solutions is 2^{40}, which is of a magnitude of trillions; and so forth. So tests are performed using different I vector lengths as is shown in Figures 6.3 and 6.4. The test is repeated

[3]When the number of intrusion is enlarged to hundreds and thousands, the problem of false negatives appears—see Section 6.2.2.1.

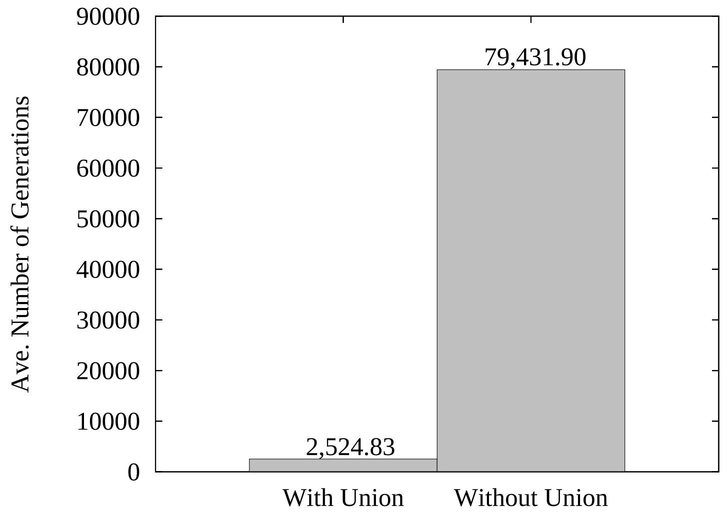


Figure 6.2: Average number of generations with union operator and without union operator; length of I: 24, 32, 40, and 48; 70 runs per length; 280 runs in total.

70 times for each length vector and different input data (Observed Vectors) as is shown on Table 6.6, where the union operator is used, and Table 6.7 where the union operator is not used. Each entry on Tables 6.6 and 6.7 corresponds to an average over 10 runs performed with the same input vector. For example, the first entry, which corresponds to user User2051_7[4] and I vector of length 24, is equal to an average of 193.1 over 10 runs. When using the union operator the maximum average number of generations needed is $1,158.8$ for a I vector with length 48, against $127,746.5$ for a vector of length 24 when not using the union operator (see Tables 6.6 and 6.7).[5]

With the fitness function as in Equation 6.3 and using the union operator, the GA did not give false positives or false negatives in the present tests. However, the problem of finding the complete set of intrusions is open and more tests are going to be done in Section 6.2.2.1 enlarging the attack-event matrix AE from 48 to 1008 intrusions.

[4]Userxxxx_y, means user with identification xxxx at time y.

[5]It should be emphasized that for I vectors of length 24, the average is quite large for the last entry, perhaps because usually the algorithm converges quicker to a local optimum, then, it spends more iterations to get a global maximum. Once the algorithm converges, the mutation operator is the one that can help the algorithm to reach the global maximum

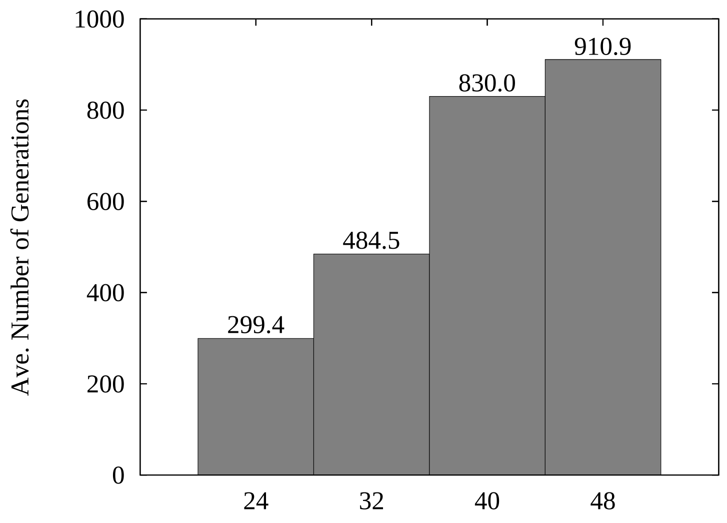


Figure 6.3: Average number of generations with union operator; length of I: 24, 32, 40, and 48; 70 runs per length; 280 runs in total.

OV	*I Length*			
	24	32	40	48
User2051_7	193.1	343.7	565.2	661.9
User2051_11	220.1	484.3	539.7	814.1
User2506_15	278.6	328.0	925.8	790.9
6 Intrusions	344.3	319.8	843.7	1,129.1
9 Intrusions	291.8	618.8	1,115.8	1,158.8
11 Intrusions	420.2	570.1	852.1	955.7
12 Intrusions	347.8	727.1	967.5	865.8

Table 6.6: Average number of generations per user/activity and I length. Union operator used.

OV	*I Length*			
	24	32	40	48
User2051_7	193.1	343.7	565.2	661.9
User2051_11	220.1	925.8	539.7	814.1
User2506_15	515.1	328.0	925.8	790.9
6 Intrusions	1,228.8	319.8	2,310.2	1,129.1
9 Intrusions	9,795.8	7,181.6	6,765.6	16,206.2
11 Intrusions	40,292.9	45,799.2	47,148.4	54,993.3
12 Intrusions	127,746.5	58,933.4	53,875.8	75,473.3

Table 6.7: Average number of generations per user/activity and I length. Union operator *not* used.

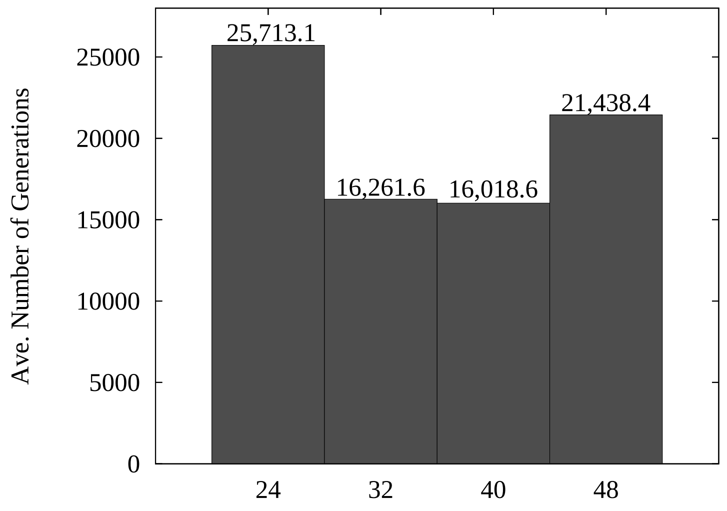


Figure 6.4: Average number of generations without union operator; length of I: 24, 32, 40, and 48; 70 runs per length; 280 runs in total.

6.2.2 Additional Contributions

For the case of the intrusion detection problem, some questions arise because of the size of the search space, i.e., if the algorithm can handle hundreds or thousands of intrusions. Some questions also arise regarding the possible solution of the problem with a different heuristic or iterative method.

6.2.2.1 Enlarging the Number of Intrusions

For the case of the intrusion detection problem, as the number of intrusions grow, the search space increase exponentially. If the number of intrusions is 48, then the search space is 2^{48}, likewise, if the number of intrusions is $1,008$, then the search space is $2^{1,008}$. So, in order to test the viability of the GA for hundreds and thousands of intrusions in the AE matrix, empirical studies is done with the following GAs' parameters and fitness function as in Equation 6.3: 60% probability of one-point crossover, 2.4% probability of mutation per chromosome, $20,000$ generations, selection pressure of 1.5 (tournament selection of size 2 with 75% of choosing the fittest), and a population size of 40. With this configuration of parameters the tests were perfomed 30 times for 48, 144, 528, and $1,008$ intrusions (Diaz-Gomez and Hougen, 2007e). As the number of intrusions in the AE matrix grow, the average percentage of false negative grows, in the present tests.

AE	*Average of False Negatives*		
Length	*40 Ind.*	*100 Ind*	*1,000 Ind*
48	0.01	0.00	0.00
144	0.14	0.11	0.02
528	0.66	0.44	0.29
1,008	0.99	0.67	0.39

Table 6.8: Enlarging the number of intrusions in the AE matrix to 48, 144, 528 and $1,008$ intrusions. False negatives appears.

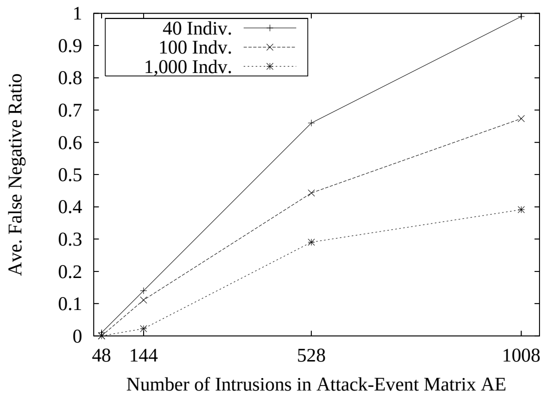


Figure 6.5: Average $\%$ of false negatives when Attack-Event Matrix AE is enlarged. Population sizes of 40, 100, and $1,000$ individuals. 30 runs on each population size.

But, as AE was enlarge to thousands of intrusions (and likewise the search space) and the number of individuals in the inital population was fixed, a parameter that could be increased in order to diminish the false negative ratio, is the population size (see Table 6.8).

If the number of individuals in the GA's population is changed from 40 to 100 and $1,000$ (keeping other parameters the same) the GA's false negative percentage is better, as expected (see Figure 6.5 for the average number of false negatives when using 40, 100 and $1,000$ individuals). However, setting parameters so that the quality of the solution is better is one of the difficulties in working with GAs.

6.2.2.2 Two Different Approaches to Misuse Detection

In order to analyze the possible benefits of the initial solution to the intrusion detection problem suggested by Mé (1993) using a GA, a comparison is made with an iterative process and with a neural network to solve the problem. It was highlighted in Section 6.2.2.1, that as the number of intrusions N_a grows, the search space grows exponentially 2^{N_a}, however, each intrusion (column) of the AE matrix could be tested against the observation vector OV in order to check if that particular intrusion occurred and, if some intrusion have been occurred, then, iteratively those possible intrusions can be tested together if there is violation of the constraint. The two sets of abnormal and abnormal exclusive could then be generated iteratively. But, this is a possible solution because the explosion of possible combinations of intrusions that could be together with and without the violation of the constraint.

The second approach is the use of a neural network to solve the problem, this work is done for the benefit of comparing different approaches.

An Iterative Process vs. A Genetic Algorithm Usually an IDS processes log records received from the operating system for a specific period of time in order to have a complete set of user activity (Bace, 2000; Crosbie and Spafford, 1995). After that, the IDS performs analysis of the current activity, using a rule base system, statistics, or a corresponding heuristic, in order to determine the possible occurrence of abnormality or intrusion.

For the present, the misuse mechanism uses a pre-defined matrix AE of intrusion patterns (Mé, 1998), so the system knows in advance the appearance of misuse and/or abuse. The bigger this table, the more space and computation time is spent in the analysis. This work continues with hundreds and more than one thousand intrusions and the same set of GA's parameters as in Section 6.2.2.1.

If a column P of the AE matrix, is such that each entry of it is less or equal than each entry of the observed vector OV, then, it is possible that intrusion P has occurred. However, looking at some possible intrusions together, it is possible that one or several can occur, but not all together, because adding each corresponding entry, some results could be greater than the corresponding entry of the user activity vector OV (Diaz-Gomez and Hougen, 2006a, 2005c; Mé, 1993). This is called a *violation of the constraint*.

The iterative process first checks each intrusion pattern P to determine if it may have occurred. For doing that, each entry of each P array is compared with the corresponding OV entry. If there is no violation of the constraint, i.e., if $P_j \leq OV_j \; \forall \; 1 \leq j \leq N_e$, where N_e is the number of activities monitored, then P is a possible intrusion. After finding all the possible intrusions, the iterative process begins to build two sets. It adds iteratively the

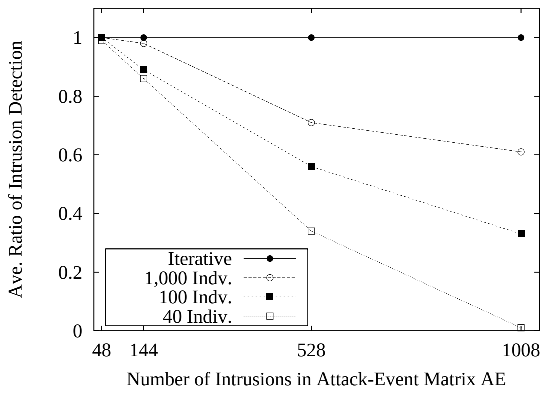


Figure 6.6: Average number of possible intrusions found by a GA and total found by an iterative process. 40, 100 and $1,000$ individuals in the initial population. 30 runs on each population size.

corresponding entries of the possible intrusions found, in order to check for a violation of the constraint when considered together. If in doing such addition, an entry of an intrusion violates the constraint, then that possible intrusion is marked as an exclusive intrusion. So, the result is going to be a set of possible intrusions that can occur together and a set of exclusive intrusions.

Figure 6.6 shows the average ratio of number of possible intrusions found by the GA and the total number of intrusions found by the iterative process. There were no false positives given by either algorithm and the iterative process had no false negatives in these tests.

As stated in Section 6.2.2.1 If the number of individuals in the GA's population is increased, then the GA's false negative percentage is decreased (see Figure 6.5.

The result of both algorithms is two sets: one of possible intrusions (Y) and the other of possible intrusions that could not happen at the same time as the previous ones because of violations of the constraint (Diaz-Gomez and Hougen, 2006a, 2005c; Mé, 1993). This is called *exclusive set* (X) (Diaz-Gomez and Hougen, 2005c). For the case of the iterative process, these two sets (Y and X) are always the same, but for the GA case, the two sets could be different because of the randomness involved in the process.

In this work, the exclusive set X was disaggregated, i.e., the algorithm continues looking for constraints until a set of disjoint sets X_1, X_2, ..., X_h is found, the union of which is the set X, so the entire set of possible intrusions is $Y \cup X_1 \cup X_2 \cup ... \cup X_h$, where $Y \cap X_1 \cap X_2 \cap ... \cap X_h = \emptyset$.

It should be emphasized that exclusive intrusions makes this an NP-Complete problem because the solution set $Y \cup X_1 \cup X_2 \cup ... \cup X_h$ is not unique. However, the algorithms presented in this section and Section 6.2.2.1 are finding one solution, not all possible solutions.

When looking at the computational complexity, the iterative process performs, for each P Column, N_e comparisons (N_e is the number of types of user activity), and as there are N_a such columns, then it spends $N_e * N_a$ computation time. For the second step of looking at the two exclusive set of intrusions, that depends on the actual number of possible intrusions found K. For each type of activity N_e, it checks the common type of activity in order to check for violations of constraints. So for this case it spends $N_e * K$ computation time, where $K < N_a$. In conclusion, the iterative process is $\mathcal{O}(N_e N_a)$.

For the case of the GA, it depends on the population size S, number of generations G and length N_e of each P column. So, for each hypothesized array I—of length N_a, and the ones that belong to the population—the algorithm performs N_a calculations for each type of activity, and as there are N_e types of activity, this gives $N_e * N_a$ calculations. As the population size is S, for each generation the algorithm performs $N_e * N_a * S$ calculations. As the algorithm has G generations, it gives a total computational complexity of order $\mathcal{O}(N_e N_a S G)$. Clearly, the GA cost is higher by $\mathcal{O}(SG)$ with the down side of a false negative ratio that depends in part on the population size (see Fig 6.5).

The computational complexity done by both algorithms in finding the disjointed sets X_1, X_2, ..., X_h of possible intrusions is $\mathcal{O}(K^2 N_e)$, where K is the cardinality of the disjointed set X.

The space complexity is such that both algorithms have to store the matrix of known intrusions and the user activity, i.e., $\mathcal{O}(N_a * N_e)$. The GA, additionally, has to store the population that is of order $\mathcal{O}(S N_a)$.

As observed, the iterative process outperformed the GA for this test set, as established by the false negative ratio and in computational and space complexity. The population size of the GA was increased in order to improve the quality of the solution—fewer false negatives—but other parameters may be changed in the GA as well, such as the number of generations and the probability of the operators, in trying to improve its performance. However, some of those possible changes may or may not improve the quality of the

21	48	69	96	117	144	165	192	213	240	261	288	309	336	357	384	405	432	453	480	501
528	549	576	597	624	645	672	693	720	741	768	789	816	837	864	885	912	933	960	981	-

Table 6.9: Intrusions I_p found by a neural network.

solution and some may expend more computation time. The correct setting of parameters is one of the difficulties in working with GAs.

A Neural Network vs. A Genetic Algorithm The misuse detection problem can be formulated as: Given the observation vector $OV \in \mathbb{Z}^{+} \cup \{0\}$, and the Attack-Event matrix $AE \in \mathbb{Z}^{mn}$ of known intrusion types, find the best parameter vector $I \in \{0, 1\}$ such that $r_j(I) = (AE * I)_j - OV_j \leq 0$, for all $0 \leq j \leq m$, where I_p are independent variables for all $0 \leq p \leq n$ (m is the number of event types to consider N_e and, n is the number of intrusions to check N_a). The *best* I is the one that minimizes the length of the residual $r(I)$, i.e., we are facing a linear least squares problem, that can be solved with different methods. However, one can look at the problem as a linear constrained optimization problem, where a Neural Network (NN) can be proposed to solve it.

How the problem is addressed can reveal different methods to solve it. Some methods require more computation time and/or space than others, and some give better quality solutions than others. This section presents a NN to solve approximately the misuse detection problem and compares it with the GA tested in Section 6.2.2.1.

Neural networks have been widely used to solve optimization problems (Ham and Kostanic, 2001) and, as was addressed previously, the misuse detection problem can be seen as an optimization problem where we want to maximize $f(I) = w^T \cdot I$, subject to $r_j(I) = AE_{j1}I_1 + AE_{j2}I_2 + ... + AE_{jn}I_n - OV_j \leq 0$ for $j = 1, 2, ..., m$, $I_1 \geq 0,\ I_2 \geq 0, ...,\ I_n \geq 0$, where I_p are independent variables and w is the weighting vector.

In order to solve this linear problem with inequality constraints, Ham and Kostanic (2001) propose the use of a NN with the recursive equation of motion

$$I_p(k+1) = \begin{cases} I_p(k) - \mu_p \left\{ w_p + K \sum_{j=1}^{m} r_j(I) AE_{jp} \right\} \\ \qquad\qquad \text{if } I_p(k+1) \geq 0, \\ 0 \qquad\qquad \text{if } I_p(k+1) < 0 \end{cases} \tag{6.4}$$

where μ_p is the learning rate, K is a positive parameter, and k is the iteration step.

The following parameters is set: $I_p(0) = 0$ for all p, $\mu_p = \mu_0/(log(1+k))$ with $\mu_0 = 0.005$ (Ham and Kostanic, 2001), $w_p = 1\ \forall p$, $K = 1$, OV—that is used in $r_j(I)$—as in Table 6.10, AE corresponds to the $m * n$ matrix in which columns are intrusions, $m = 28$, $n = 1,008$, and the NN stops if $\mu_p < 0.00001$ or if the number of iterations is $h = 6,000$.

The net found 41 out of 108 possible intrusions (see Table 6.9) and had no false positives. Some convergence values for iterations until 600 are shown in Figure 6.7.[6] The last μ_p was $\mu_{6000} = 0.00057473$.

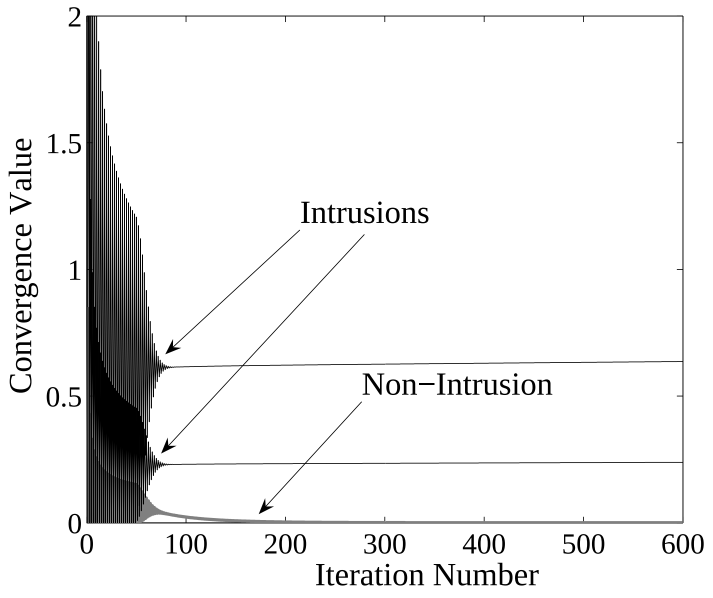


Figure 6.7: Intrusions type 48 and 21 found by a neural network. At iteration $6,000$ the convergence values were $I_{48} = 0.6426$, $I_{21} = 0.241$ and non intrusion $I_{917} = 3.5889e-34$.

Table 6.10 shows an example of an OV vector and the result of $AE*I-OV$ that shows that the neural network found an I vector that does not violates the constraint. The solution I is such that $I_p \geq 0,\ \forall p$. For example, looking at entries in the AE matrix, intrusions $p = 21 + mod(0,48)$ have values $AE_{26,21+mod(0,48)} = 3$ and there were 21 of those; intrusions $p = 48 + mod(0,48)$ have values $AE_{26,48+mod(0,48)} = 8$ and there were 20 of those; they give a total of activity for entry 26 equal to $3*21*0.241+8*20*0.6426 = 118$ that is exactly OV_{26}—see Figure6.7 and Table 6.10.

[6]Tendency after iteration 600 is the same

$Event\ \#$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
OV	0	0	0	0	0	0	0	1	0	0	5	0	0	0	0	1	0	25	0	13	0	0	0	2	0	118	315	0
$AE * I - OV$	0	0	0	0	0	0	0	-1	0	0	-5	0	0	0	0	-1	0	-25	0	-13	0	0	0	-2	0	0	-315	0

Table 6.10: Event type, vector of observations OV and constraint comparison using solution I—shown in Table 6.9—which does not violate the constraint.

S_1	21	69	155	157	192	213	261	309	336	384	453	480	501	528	576	683	693	741
S_1	768	789	837	864	933	960	1008	-	-	-	-	-	-	-	-	-	-	-
S_2	48	96	117	144	165	240	288	357	405	432	549	597	623	644	815	884	911	980

Table 6.11: A Subset of intrusions S_2 that violates the constraint with subset S_1 found by a GA.

It should be emphasized that if the initial conditions change, for example if $I_p(0) = 1$ for all p, then the algorithm converges to a second solution. It finds all possible solutions (108), but, in this case the solution violates the constraint and it gives 399 false positives (Diaz-Gomez and Hougen, 2007d).

The first topic that we are going to address is how the algorithms presented here, distinguish an intrusion of a non intrusion and the second one is the computational complexity of each algorithm.

For the GA it is clear that a 1 in I_p means a possible intrusion p occurred and a 0 means non-intrusion. For the NN, if I_p converges to a value > 0 then we consider a possible occurrence of an intrusion. However, there is not an exact threshold for the NN to distinguish an intrusion from a non intrusion, as in the GA case. In order to reinforce this fact, we performed tests again, with the same set of parameters defined in section 6.2.2.2 but the vector of observations OV was changed (OV') as in Table 6.12. The solution of the NN was the same in section 6.2.2.2—see Table 6.9—but the values of convergence of the intrusions ($I_{48} = 0.0163$, $I_{21} = 0.0061$) and non intrusion changed ($I_{917} = 0$). The NN was looking at each variable (intrusion) I_i independently, as it is expected to do in accordance with the conditions of this paradigm—the most that concerns here is the convergence of $I_p \geq 0$ and that I_p's are independent. To the contrary, the GA looks the possible solution I, with all its components I_p, together, i.e., if there is a possible solution I which violates the constraint, then, I is penalized accordingly—see Equation 6.3. The set of I_p's are evaluated by the GA at the same time and the algorithm chooses them by looking for the best of those sets (Diaz-Gomez and Hougen, 2007d).

OV'	0	0	0	0	0	0	1	40	0	0	5	0	0	0	0	1	0	25	0	13	0	0	0	2	0	3	30	0
$AE * x - OV'$	0	0	0	0	0	0	-6	-40	0	0	-5	0	0	0	0	-1	0	-25	0	-13	0	0	0	-2	0	0	-30	0

Table 6.12: Vector of observations OV'. Same Solution I—shown in Table 6.9—which does not violate the constraint.

S_1	21	48	69	96	117	144	165	192	213	240	261	288	309	336	357	384	405	432	453	480	501	549
S_2	528	576	597	624	645	672	693	720	741	768	789	816	837	864	885	912	933	960	981	-	-	-

Table 6.13: Second Phase. A Subset of intrusions S_2 that violates constraint with subset S_1 found by iterative process.

As NN does not have the capability to look at exclusive sets of intrusions ($S_1 \cap S_2 = \emptyset$), because I_p are independent for $0 \leq p \leq n$, an iterative process that receives as input the output of the NN—i.e., Table 6.9—and analyzes violations of constraint using $I_p \in \{0, 1\}$ can be used as a second phase. This process looks at each row of the AE matrix for columns corresponding to the positions of the NN solution I where I_p is considered a possible intrusion. The output is a subset of Table 6.9 given in Table 6.13 as S_1. This time we obtain 12 intrusions type $21 + mod(0, 48)$—see Section 6.2.2.2—and 10 intrusions type $48 + mod(0, 48)$, which gives us a total of $3 * 12 * 1 + 8 * 10 * 1 = 116$ which clearly does not violate the constraint (i.e. $116 \leq OV_{26} = 118$). More than this 22, will begin to violate the constraint—see S_2 in Table 6.13 (Diaz-Gomez and Hougen, 2007d).

The NN needs to calculate the constraint, i.e., $AE * I - OV$ which has a cost of $m * n$, it adjusts I and, as the algorithm iterates h times, it gives an estimated computational complexity of $\mathcal{O}(mnh)$. The GA needs to calculate the constraint for each individual in the population that has a cost of $m*n$ per individual, i.e., with s individuals it gives $m*n*s$ per generation, and as the algorithm iterates g generations, it gives a total computational complexity of $\mathcal{O}(mnsg)$ (Diaz-Gomez and Hougen, 2007e). So the GA computational complexity is higher by $\mathcal{O}(sg/h)$.

The space complexity for the NN can be considered as $\mathcal{O}(nm)$ because it needs to store the AE matrix, and the OV and I vectors. The GA, besides previous structures, needs to store the population that is of order $\mathcal{O}(sl)$. So the GA space complexity is higher in $\mathcal{O}(sl)$ than the NN space complexity.

Comparing these two paradigms, as some intrusions share the same types of events, the possible solution I is such that some I_p are dependent, which makes the genetic algorithm paradigm more suited for solving this problem. However, the quality of the solution obtained with the GA has a higher computational complexity cost of $\mathcal{O}(sg/h)$—population size by the ratio of number of generations over the NN iterations—and space complexity cost of $\mathcal{O}(sl)$—population size by length of I—with respect to the NN.

The GA has the advantage of discriminating an intrusion from a non-intrusion as the solution of the problem is encoded as 1 (intrusion) and 0 (non-intrusion). As the range of values of I_p for the NN are such that $I_p \geq 0$ the values of intrusions are input dependent—depending on the observed vector OV. However, at least for this test set, non-intrusions are variables I_p that converge to 0 or to values ≈ 0 when in the initial conditions I is zero.

For the test set defined in this section, there were no false positives, except if the NN is considered without the second phase or if the initial conditions change. For the false negative side, if we look at the two sets S_1 and S_2, the GA has in average (over 30 runs) of 39.14% false negatives, and the NN has 60.95%. However, the set S_2 can have exclusive intrusions, so the process can continue until we get a set of mutually exclusive subsets whose union is S (Diaz-Gomez and Hougen, 2007e).

In order to improve the false negative ratio of the GA, it is possible that by increasing the population size ($s > 1,000$) the ratio is going to decrease; however, it is possible that the number of generations g should be considered too, independently or in conjunction with the population size. For the case of the NN, it is a more challenging problem to try to diminish the false negative ratio. After the convergence of all I_p's there is no improvement in the solution I, if the number of iterations h is higher.

6.2.3 Fitness Functions to Hunt Snakes in Hypercubes

In this section we are going to report our work in the snake-in-the box problem, which is of particular interest in coding and hypercube-based computing and networking, using a standard GA, i.e., using binary encoding, 2-tournament selection, one point crossover with a probability of 0.6, a mutation rate of 0.024 per chromosome, a population size of $1,000$, and as stop criteria $1,000$ generations, or when the algorithm finds a snake. We propose seven fitness functions for hunting snakes, and we perform initial test in hypercubes of dimension 4, in order to evaluate their effectiveness.

Node	*Adjacency Matrix* (*AM*)								S	VN
0	0	1	1	0	1	0	0	0	1	$\underline{1}$
1	1	0	0	1	0	1	0	0	1	$\underline{2}$
2	1	0	0	1	0	0	1	0	0	3
3	0	1	1	0	0	0	0	1	1	$\underline{2}$
4	1	0	0	0	0	1	1	0	0	2
5	0	1	0	0	1	0	0	1	0	2
6	0	0	1	0	1	0	0	1	1	$\underline{1}$
7	0	0	0	1	0	1	1	0	1	$\underline{2}$

Table 6.14: Adjacency Matrix (*AM*) for a 3-dimensional hypercube, Snake (S) $0 - 1 - 3 - 7 - 6$, and Vector of Neighbors (*VN*).

The snake was encoded as an uni-dimensional array (chromosome) of length 2^d, where d is the dimension of the hypercube in which to search for snakes. To encode the hypercube itself, an adjacency-matrix representation is used. To calculate the number of neighbors of each node in a snake, the matrix multiplication between the adjacency-matrix and the snake is performed. Table 6.14 shows the nodes $\{0, 1, 3, 6, 7\}$ in the array S, which may be seen as the connected path $0 - 1 - 3 - 7 - 6$, with nodes $1, 3$, and 7 having two neighbors in the path, and nodes 0 and 6 being the nodes with one neighbor each. It should be noted that node 2 has three neighbors, but that node does not belong to the path S, so its value in S is 0 (Diaz-Gomez and Hougen, 2006b).

Some fitness functions have been suggested for hunting snakes in d-dimensional hypercubes (see Section 5.1.3), from Equation 5.17 to Equation 5.23 that are going to be rewritten here for simplicity). Results in a 4-dimensional hypercube and initially with at most $1,000$ generations, are summarized in Table 6.15, where each test was made over 30 runs to obtain more statistically interesting results. Equation 6.5, followed by Equations 6.10 and 6.6, found more snakes, but, of the three, Equation 6.6 finds more of maximum length. Equation 6.8 is the only one that has bad points, i.e., points that violate the constraint having more than two neighbors (14 in total in two out of the 30 runs) and, as addressed in Section 5.1.3.4, that Equation is deviating the algorithm.[7] All these results are obtained when the algorithm is stopped when it finds a maximum (local or global)

[7]In Table 6.15 Equation 6.8 obtained 60 end points ($\#D$), however this does not mean that using that equation in all the 30 runs the algorithm reported one head and one tail. For this Equation, one time the algorithm found a solution with four end points and in two solutions the algorithm found a solution with one end point. For the rest of Equations the algorithm found two end points in all the 30 runs.

Eq.	*# Snakes*	*# Longest*	*# T.Isolated*	*# T.Lazy*	*# T.Bad*	*#D.*
6.5	30	5	0	3	0	60
6.6	27	24	3	0	0	60
6.7	24	15	6	4	0	60
6.8	21	19	7	0	14	60
6.9	21	20	9	0	0	60
6.10	28	16	2	7	0	60
6.11	26	16	4	4	0	60

Table 6.15: Results of snakes in a 4-dimensional hypercube; different fitness functions; $\leq 1,000$ generations; 30 runs.

in less than $1,000$ generations or when $1,000$ generations is reached (Diaz-Gomez and Hougen, 2006b).

$$F(I) = \left(\frac{\sum_{j=0}^{2^d-1} (AM * S)_j - Penalty}{\sum_{j=0}^{2^d-1} (AM * S)_j} \right) \left(\frac{Length(S) + 1}{\#P} \right). \tag{6.5}$$

$$F(I) = \left(\frac{\sum_{j=0}^{2^d-1} (AM * S)_j - Penalty}{\sum_{j=0}^{2^d-1} (AM * S)_j} \right) * Length(S). \tag{6.6}$$

$$F(I) = Length(S). \tag{6.7}$$

$$F(I) = (\#P - \#Lazy) * Length(S). \tag{6.8}$$

$$F(I) = Length(S) - \#Lazy. \tag{6.9}$$

$$F(I) = Length(S) - \#Lazy - \#Isolated \tag{6.10}$$

$$F(I) = \frac{Length(S)}{1 + Penalty} \tag{6.11}$$

	Equation					
Eq.	6.5	6.6	6.7	6.8	6.9	6.10
6.6	0.005	-	-	-	-	-
6.7	0.000	0.055	-	-	-	-
6.8	0.026	0.537	0.055	-	-	-
6.9	0.002	0.760	0.109	0.936	-	-
6.10	0.000	0.109	0.760	0.342	0.760	-
6.11	0.000	0.109	0.342	0.537	0.537	0.887

Table 6.16: K-S test results for snakes. Probability of results assuming the null hypothesis. $\leq 1,000$ generations. 30 runs per fitness function.

	Number of Generations			
Fitness Function	*Minimum*	*Maximum*	*Average*	σ
Eq. 6.5	26	$17,032$	$2,874.2$	$4,104.0$
Eq. 6.6	6	$25,352$	$3,174.1$	$5,414.7$
Eq. 6.7	8	$20,435$	$2,044.4$	$4,187.6$
Eq. 6.8	2	$9,807$	$2,739.6$	$3,032.2$
Eq 6.9	4	$15,138$	$2,770.3$	$3,939.4$
Eq. 6.10	1	$13,089$	$1,750.5$	$3,015.5$
Eq. 6.11	5	$17,117$	$2,295.5$	$4,206.4$

Table 6.17: Results of number of generations for hunting longest snakes in a 4-dimensional hypercube; different fitness functions; 30 runs.

Table 6.16 shows the K-S test results, for the # of snakes as in Table 6.15, for these equations when a maximum number of generations of $1,000$ is used. Basically Equation 6.5 is different from most of the others in finding snakes in 4-dimensional hypercubes, as is shown in Table 6.15.[8] This is also reflected in the histograms as in figures 6.8 to 6.14 where it can be seen that the data is not normally distributed (fact that suggests the realization of the K-S test which makes no assumption of the distribution[9]).

But how would the results turn out if the algorithm is freed to find the longest snake, not only a snake when the number of generations is $\leq 1,000$, i.e., when the stopping criteria is the finding of the longest snake—in this particular analysis when length is 7.

Table 6.17 shows the minimum, maximum, average (mean), and standard deviation σ of the number of generations. As can be observed, the standard deviation is greater than the average in all cases which suggest that the distribution is not normally distributed

[8]There is a multiple comparison problem here, that we do not address, because this is an exploratory study that allow us to get an idea of how these fitness functions work.

[9]*Frequency* corresponds to ne number of snakes as is Table 6.15.

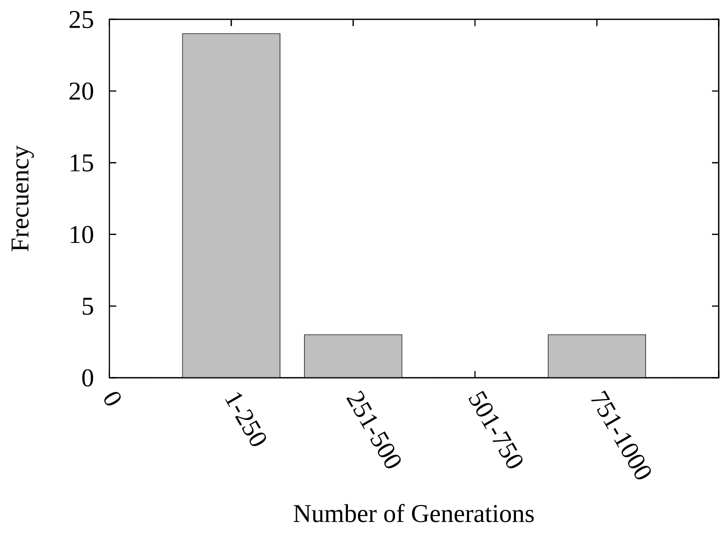


Figure 6.8: Histogram for hunting snakes using Equation 6.5; generations $\leq 1,000$.

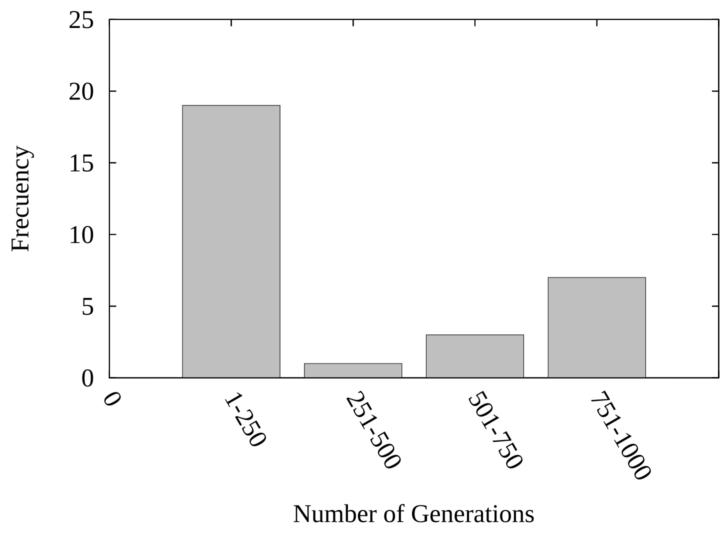


Figure 6.9: Histogram for hunting snakes using Equation 6.6; generations $\leq 1,000$.

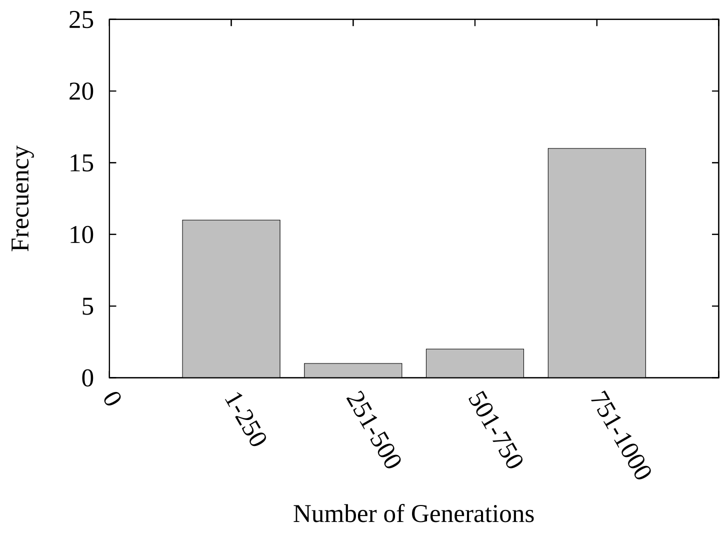


Figure 6.10: Histogram for hunting snakes using Equation 6.7; generations $\leq 1,000$.

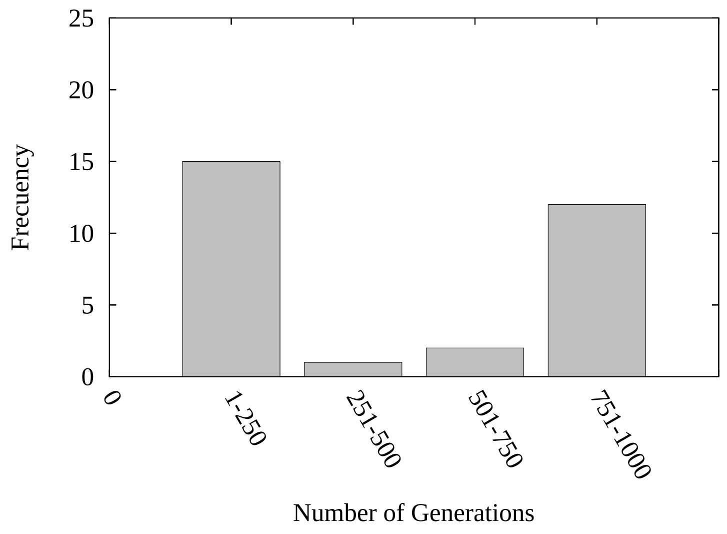


Figure 6.11: Histogram for hunting snakes using Equation 6.8; generations $\leq 1,000$.

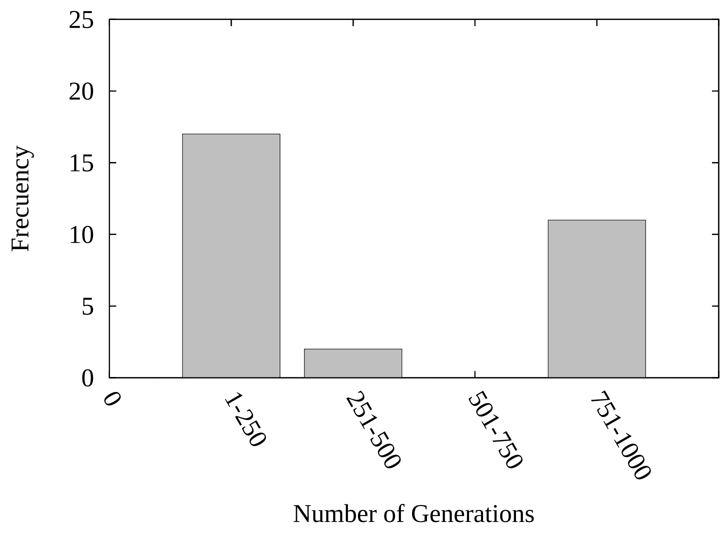


Figure 6.12: Histogram for hunting snakes using Equation 6.9; generations $\leq 1,000$.

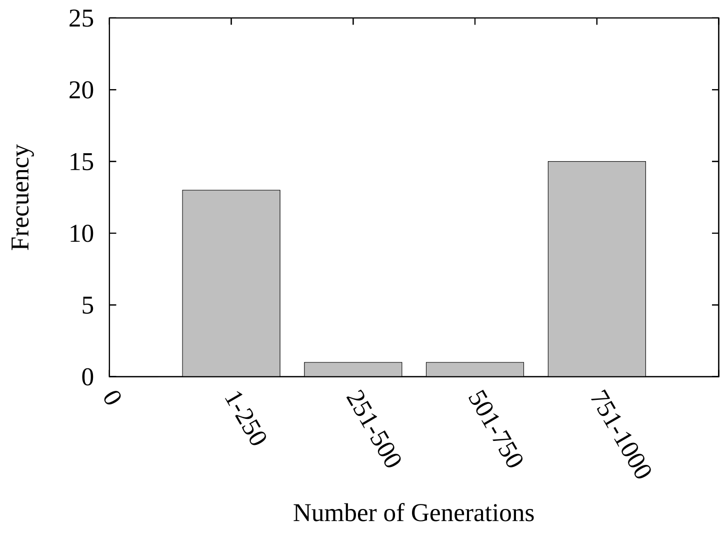


Figure 6.13: Histogram for hunting snakes using Equation 6.10; generations $\leq 1,000$.

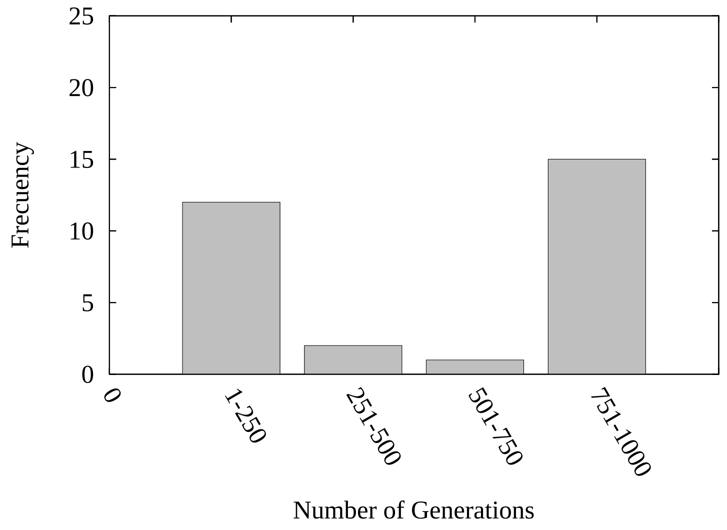


Figure 6.14: Histogram for hunting snakes using Equation 6.11; generations $\leq 1,000$.

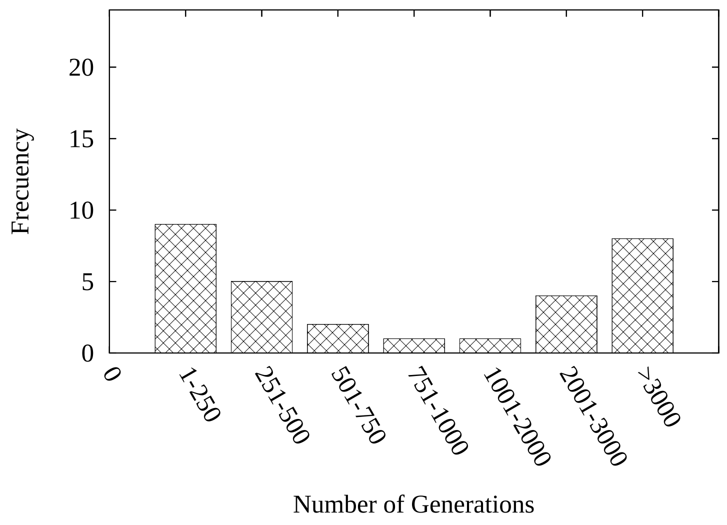


Figure 6.15: Histogram for longest snakes using Equation 6.5; generations until global maximum reached.

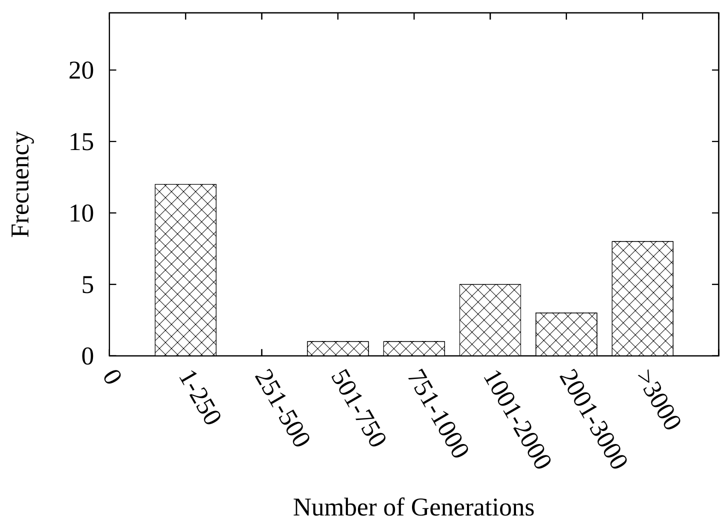


Figure 6.16: Histogram for longest snakes using Equation 6.6; generations until global maximum reached.

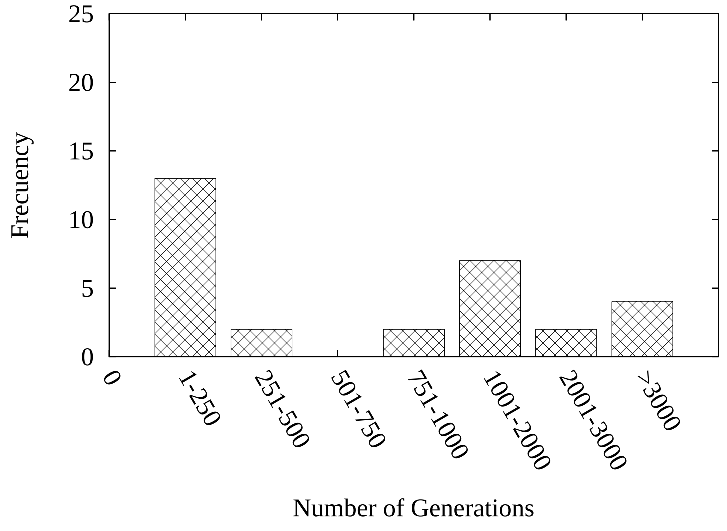


Figure 6.17: Histogram for longest snakes using Equation 6.7; generations until global maximum reached.

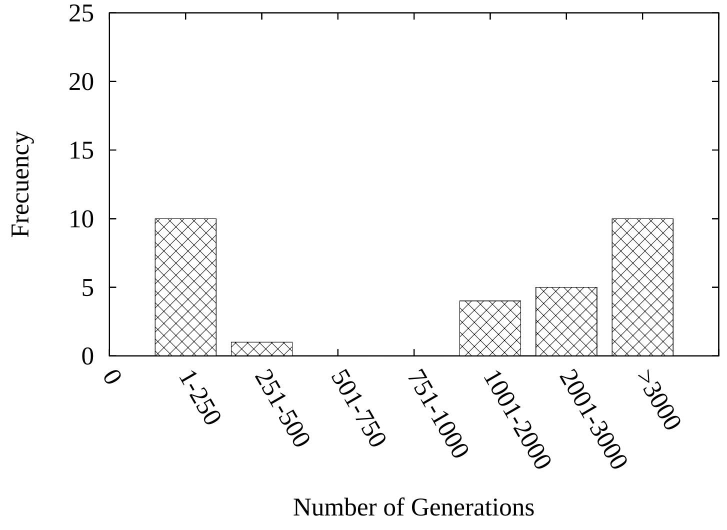


Figure 6.18: Histogram for longest snakes using Equation 6.8; generations until global maximum reached.

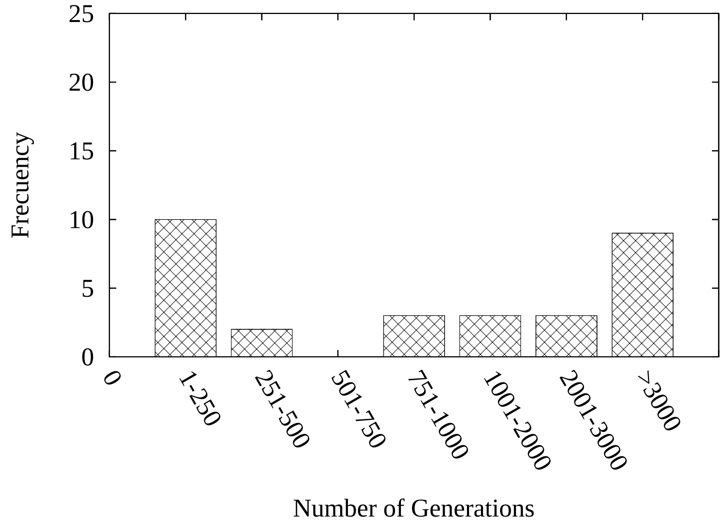


Figure 6.19: Histogram for longest snakes using Equation 6.9; generations until global maximum reached.

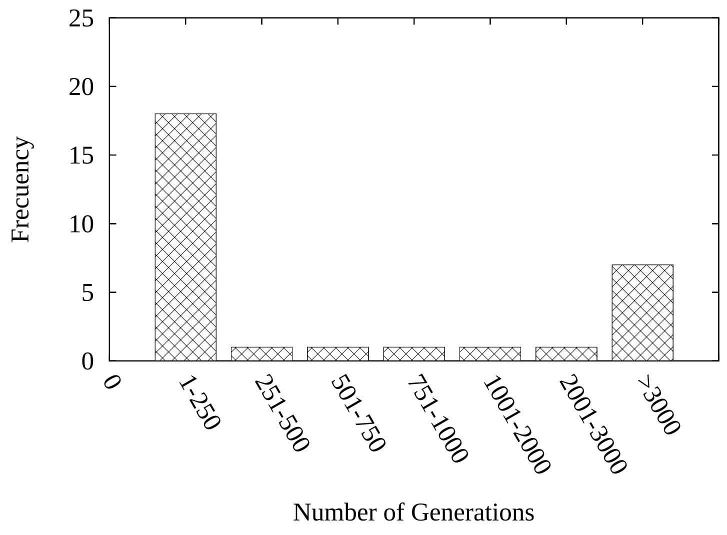


Figure 6.20: Histogram for longest snakes using Equation 6.10; generations until global maximum reached.

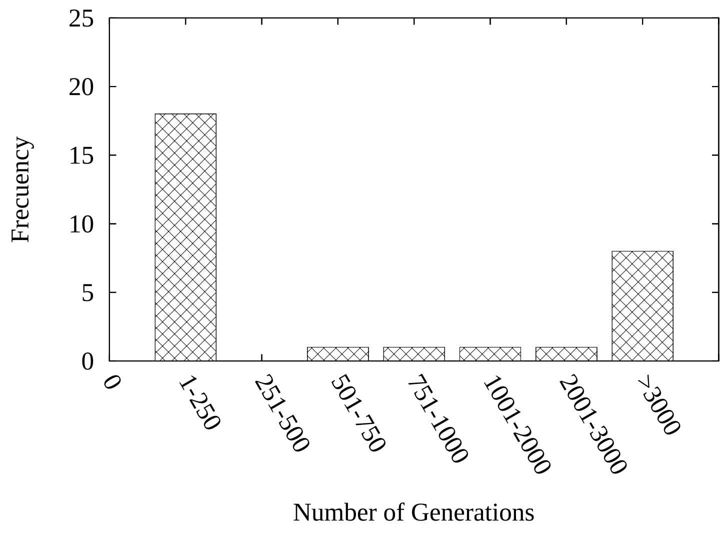


Figure 6.21: Histogram for longest snakes using Equation 6.11; generations until global maximum reached.

	Generations until Longest Found					
Eq.	6.5	6.6	6.7	6.8	6.9	6.10
6.6	0.200	-	-	-	-	-
6.7	0.342	0.537	-	-	-	-
6.8	0.537	0.936	0.109	-	-	-
6.9	0.537	0.537	0.537	0.537	-	-
6.10	0.011	0.200	0.342	0.055	0.055	-
6.11	0.011	0.342	0.537	0.109	0.200	0.997

Table 6.18: K-S test results for longest snakes. Probability of results assuming the null hypothesis. Number of generations until longest snake found. 30 runs per fitness function.

	Equation						
Quartil	6.5	6.6	6.7	6.8	6.9	6.10	6.11
$Q1$	205.0	45.5	93.0	126.0	839.5	46.0	44.0
$Q2$	539.5	$1,059.0$	671.0	$1,827.0$	$1,238.5$	161.5	151.0
$Q3$	$5,708.0$	$3,979.5$	$1,969.5$	$4,867.5$	$4,569.0$	$3,596.5$	$3,389.5$

Table 6.19: Quartiles for fitness functions to hunt snakes. Number of generations until longest snake found. 30 runs per fitness function.

because the minimum is zero (see histograms in Figures 6.15 to 6.21). There is no statistically significant difference for this test set for any fitness function (except between Equation 6.5 and Equations 6.10 and 6.11) as can be seen in the results presented in Table 6.18.

Table 6.19 shows the quartiles for different fitness functions presented in this section in finding longest snakes, over a total of 30 runs for each one. For this test set, Equation 6.7 outperformed others in the number of generations spend in finding longest snakes. It can be observed that 75% Equation 6.7 spent less than $1,969.5$ generations for finding longest snakes. For this data set (using Equation 6.7), there were three outliers, the data points $20,435$, $10,751$ and $7,773$ (which makes its mean, presented in Table 6.17, biased) because those are out of the $1.5 * IQR + Q_3$ where $IQR = Q_3 - Q_1$ (Mann, 2007). Now, taking for example Equation 6.5, 75% of the times the algorithm spent less than $5,708.0$ generations to find a longest snake.

Our analysis lead us to belief that Equation 6.7 is the appropriate fitness function to use in some of the test performed in Chapter 7.

Equation	*Seed*							
	0	1	2	3	4	5	6	7
6.5	$2,874.2$	719.7	$1,647.4$	$1,578.3$	$1,921.6$	$1,002.4$	798.2	$2,285.3$
6.6	$3,174.1$	1850.9	$3,116.7$	$1,713.3$	$2,274.3$	$1,834.0$	$1,922.1$	$2,222.8$
6.7	$2,971.8$	2208.0	$2,494.7$	$1,685.8$	$1,541.0$	$2,194.1$	$2,600.8$	$2,083.3$
6.8	$3,356.4$	3830.1	$3,114.9$	$1,555.6$	$2,982.7$	$1,169.7$	$1,978.7$	$1,511.5$
6.9	$1,346.4$	1600.5	$1,799.6$	$3,063.9$	813.4	922.9	$1,947.1$	$1,547.8$
6.10	$1,452.7$	1474.1	$2,045.0$	$2,308.8$	$1,804.9$	870.5	$1,215.8$	$1,701.6$
6.11	$3,264.2$	2704.9	$2,621.0$	$2,006.5$	$1,573.9$	$1,585.2$	$3,086.4$	$1,812.4$
Totals								
FirstEval	$18,437.8$	$14,388.2$	$16,839.4$	$13,912.1$	$12,911.7$	$9,578.8$	$13,549.0$	$13,164.5$
SecondEval	$12,418.9$	$13,407.1$	$9,697.6$	$15,221.9$	$12,764.7$	$9,221.6$	$9,670.8$	$10,244.9$

Table 6.20: Number of generations for different fitness functions in finding longest snakes in a 4-dimensional hypercube. Initial population randomly generated with different seeds. 30 runs per seed for each fitness function.

6.2.4 Additional Contributions

For the snake-in-the-box problem, an interesting topic of investigation is how to find the length of a snake. Our method needs a head/tail to begin with; however, if the chromosome has more than two heads/tails then the algorithm could take the first head and tail and calculate the length or, as we proposed, the algorithm can look for all the possible lengths and assign the longer as the length of the snake.

6.2.4.1 Snakes' Fitness Evaluation

As stated in Section 5.1.4.2 the starting point chosen to begin the evaluation of the length of a snake, can give different results in the calculation of the length. Two types of tests were performed: in the first one *FirstEval* (see Figure 6.22), the evaluation was performed taking into account the first head and tail found, i.e., the ones with lower numeric node labels; in the second one *SecondEval*, all the distinguishing points (head/tail) are taken into account ranking all the connected paths found that obeys constraints, compares them, and takes the longest one as the actual length for the chromosome.

For this empirical study, 8 different initial population are chosen (seeds 0 through 7) and, seven fitness function for hunting snakes in 4-dimensional hypercubes are tested (see Section 6.2.3). Figure 6.22 compares the two ways to conduct fitness evaluations. With

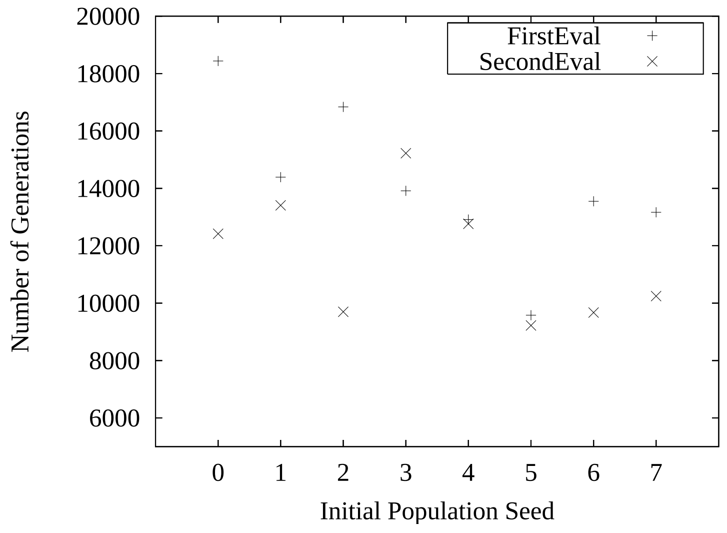


Figure 6.22: Comparison of accumulative average number of generations as in Table 6.20 ranking chromosomes in two ways: considering the first length encountered (*FirstEval*) and taking the maximum length of all encountered (*SecondEval*). Seeds values 0 through 7 shown. 30 runs per seed for each of seven fitness functions.

SecondEval there is an average reduction of 18% in the total number of generations for finding the longest snakes.

6.2.5 Conclusions

Our theory developed around the setting of some fitness functions using only information from the problem (see Section 5.1). The empirical study performed to test the second research statement regarding the better performance of some GAs when the objective(s) and the constraint(s) are joined with parameters that belong to the problem itself shows the benefit of avoiding external tuning. Section 6.2 compares a fitness function with external parameters with two functions proposed by us. The ones that need no external tuning outperform the one that needs external tuning. Results showed that increasing or decreasing an external parameter causes the algorithm to diminish or augment false positives, but pays the cost of increasing or decreasing the false negative ratio. The setting of fitness functions' parameters constitutes an optimization problem by itself. This problem is approached by trying to join the objective(s) with the constraint(s) using information from the problem. However, as some of the fitness functions proposed are normalized, they make the fitness function quite effective for finding local maxima or minima. In order to avoid this problem the use of the union operator is proposed, for the particular case of finding intrusions and, for finding snakes we propose the use of the term $length(S)$.

For the case of the intrusion detection problem, when the Attack-Event matrix, AE, was enlarged it was not enough to use the dynamic fitness function as in Equation 6.3. Nor was the use of the union operator adequate to solve the problem of false negatives. An increase in the population size was necessary in order to diminish the false negative ratio presented. This shows the conjunction of parameters in GAs and how difficult trying to improve the solution of a problem by just isolating one parameter can be.

Seven fitness functions that use parameters from the problem itself to hunt snakes in hypercubes have been tested. The first set of tests (over 30 runs) used as stop criteria the finding of a maximum (local or global) or at most $1,000$ generations. The second set of tests used as stop criteria the finding of a global maximum. For the first set of tests, the fitness functions that showed better performance in quality of the solution were Equations 6.5, 6.10 and 6.11. These functions were those that had less violations of the constraints (see Table 6.15). Equation 6.5 has no violation of the constraint,[10] Equation 6.10 has two isolated and three lazy points, and Equation 6.11 has two isolated and four lazy points. Of the three, Equation 6.10 is the one that finds the longest snakes: 21 in total, against 16 in Equation 6.11 and 2 in Equation 6.5. For the second set of tests, i.e., when the stop criteria was the finding of a longest snake, it was observed that, on average, the best fitness functions for finding longer snakes are Equations 6.9 and 6.10 (see Tables 6.17 and 6.19). Both have the peculiarity that they are linear and both join the objective (the length) with the constraint (the snake). As a matter of fact, all the fitness functions presented tried to do that, but in different ways (Diaz-Gomez and Hougen, 2007b).

Additionally, this section shows two approaches to the intrusion detection problem and the particular benefits of each one (Diaz-Gomez and Hougen, 2007d,f) in a effort to compare the GA and enhance the number of intrusions considered. Besides this, for the case of the snake-in-the-box problem, an additional contribution was presented regarding the evaluation of the length of snakes, where all the possible sub-snakes are considered, and the maximum length of those (if present) is the length that is reported as the actual length of the path.

[10]Besides the fact that Equation 6.5 has 5 lazy points, a lazy point actually does not violates the constraint, however, it is possible that it helps in finding longer snakes.

6.3 Crossover Rate vs. Mutation Rate

For years the schema theorem has been a principal mathematical foundation for Genetic Algorithms. This section performs empirical studies about the schema theorem trade-off with respect to crossover and mutation rates presented in Section 5.2.1. The one-max function, a variant of it, and finding snakes in hypercubes are used to demonstrate this trade-off, looking at the impact on solution quality and number of function evaluations required when the schema theorem trade-off is violated.

6.3.1 The One-Max Function

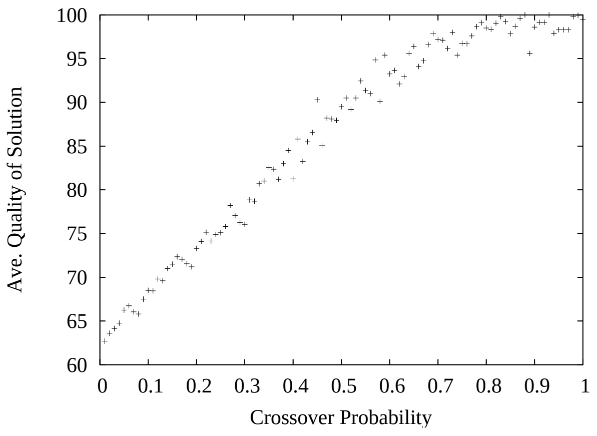


Figure 6.23: Average solution quality. 20 runs per point. $0.0 \leq p_c \leq 1.0$, $S_p = 2$, $p_m = 0.0$.

To study the schema theorem trade off, the one-max problem is presented here as a problem for which schema can be considered small because the building block is just each gene of the structure under consideration. Harik and Lobo (1999) perform tests on the one-max function reporting on average $2,500$ function evaluations over 20 different runs, using a population size of 100 structures, with $l = 100$, $S_p = 2$, uniform crossover with $p_c = 1$, and $p_m = 0$. Note that the parameters given conform to the equations that was shown in Section 5.2.1 for short schema. However, it could be the case that for some population sizes the algorithm does not converge to the optimal solution because $p_m = 0$, i.e., if the correct value of all alleles at a specific locus in the entire population are lost, then the background operator is not there to recover it (Holland, 1992).

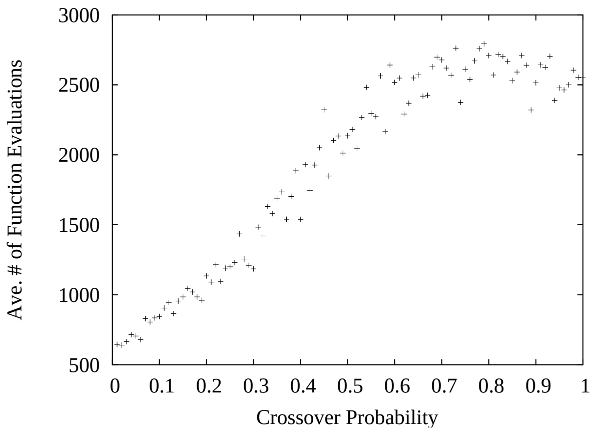


Figure 6.24: Average function evaluations. $\mathbf{20}$ runs per point. $\boldsymbol{0.0 \leq p_c \leq 1.0}$, $\boldsymbol{S_p = 2}$, $\boldsymbol{p_m = 0.0}$.

To compare with Harik and Lobo (1999), tests were performed with the one-max function using their parameters but varying the crossover probability, using a different random seed in each of 20 runs. Figures 6.23 and 6.24 show the average quality of the solution (how close it comes to 100 1's) and the average of the number of function evaluations over the 20 runs when the probability of crossover varies from 0.0 to 1.0 in steps of 0.025 and $p_m = 0.0$. As expected, the quality of the solution for this problem is better as $p_c \to 1$ (high). Two stop criteria were used: (1) If the GA reaches the maximum or (2) if at least one locus has the value 0 for the entire population, in which case the algorithm will not converge because of the missing mutation operator.

Tests were also performed varying p_m from 0.001 to 0.015 in steps of 0.001, in order to see how the performance of the algorithm changes while the other parameters and operators remain the same. This time the test was performed over 30 runs in order to look for statistical significance. The same two stop criteria were used, i.e., the algorithm reaches the maximum or at least one locus has the value 0 for the entire population. Of course, with mutation the algorithm usually can "repair" a lost gene; however, mutation can "damage" a gene too. Figure 6.25 shows the effect of mutation on the quality of the solution when p_c varies from 0.0 to 1.0 as in the previous case. Table 6.21 shows the improvement of the GA, in the average quality of the solution, when a mutation ratio of 0.015 is included in the procedure (last row). It can be observed that using selection pressure alone (first row), it is obtained on average the less solution quality; that entry gives an

procedure τ			*Qual.*		*Func. Ev.*		*Mutations*	
S_p	p_c	p_m	*Ave.*	*std.*	*Ave.*	*std.*	*Ave.*	*std.*
2	0	0.000	61.8	2.00	650.0	73.1	0.0	0.0
2	0	0.015	79.4	7.54	7,210.0	9,276.4	10,801.4	13,892.9
2	1	0.000	99.7	1.40	2,521.9	222.9	0.0	0.0
2	1	0.015	100.0	0.00	9,305.7	4,037.58	14,049.4	6,087.4

Table 6.21: Improvement of the one-max GA in solution quality and cost when crossover and/or mutation are included in procedure $\boldsymbol{\tau}$. 30 runs.

average quality of the solution equal to 61.8. At least one of the operators should be used in order to improve. Nonetheless, selection pressure is also necessary. Table 6.22 shows that with no selection pressure (i.e., $S_p = 1$) there is no improvement at all. Figure 6.26 shows all the effects: the bottom of the figure shows the use of the operators with no selection, in which case the maximum quality was 62.16 over 30 runs; the upper part shows the effect of the selection pressure and both operators. The improvement in the quality using the two operators and a selection pressure of $S_p = 2$ is evident.

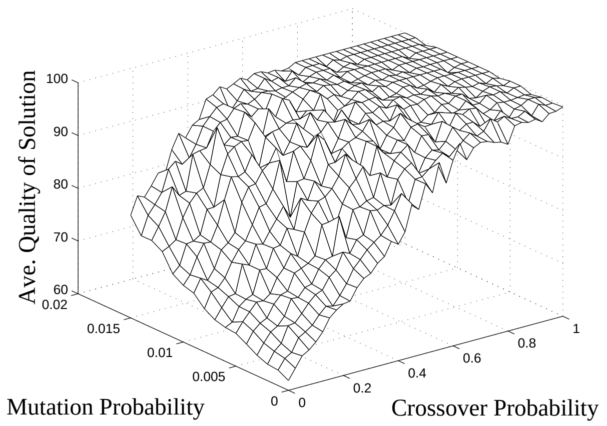


Figure 6.25: Average solution quality. $\boldsymbol{S_p = 2}$. One-max function. $\mathbf{30}$ runs per set of parameters.

6.3.2 One-Max Variant

To see if previous results generalize, a local maximum was added to the one-max function. The new function as in Figure 6.27 is

procedure τ			*Qual.*		*Func. Ev.*		*Mutations*	
S_p	p_c	p_m	*Ave.*	*std.*	*Ave.*	*std.*	*Ave.*	*std.*
1	0	0.000	56.8	2.70	3,156.6	936.1	0.0	0.0
1	0	0.015	59.8	3.33	53,940.0	50,916.7	81,019.6	76,456.7
1	1	0.000	60.5	2.20	2,336.6	569.6	0.0	0.0
1	1	0.015	60.7	1.80	50,956.6	34,626.1	76,515.8	51,970.1

Table 6.22: Improvement of the one-max GA in solution quality with no selection pressure.

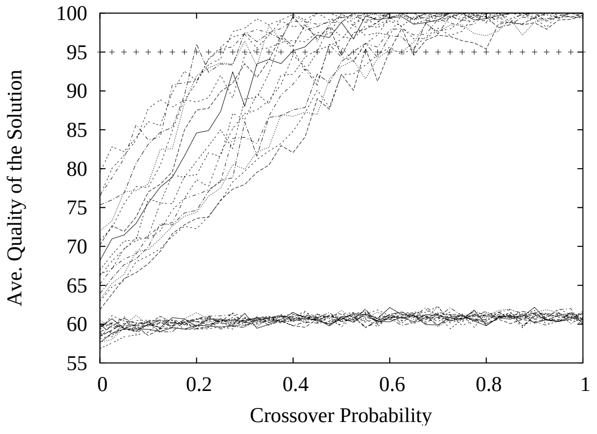


Figure 6.26: Average solution quality. $\boldsymbol{S_p = 1}$ bottom, $\boldsymbol{S_p = 2}$ top. $\boldsymbol{0.0 \leq p_m \leq 0.015}$, step size $\boldsymbol{0.001}$. Each line shows a $\boldsymbol{p_m}$ value. One-max function. $\boldsymbol{30}$ runs per set of parameters.

$$\begin{aligned} F(I) &= -0.5 * \#Ones + 75, \text{ if } \#Ones < 50 \\ F(I) &= \#Ones, \text{ otherwise.} \end{aligned} \tag{6.12}$$

The average quality obtained for this function has the same tendency as for one-max when $S_p = 2$ is used.

To see the trade-off of the schema theorem for one-max and Equation 6.12, consider the cost of different plans in reaching a solution quality of approximately 95.0. Figure 6.26 has a straight row of $+$'s at that value. All points that cross that row correspond to plans for which the average solution quality is approximately equal to 95.0; however, the cost to achieve that quality is different for each plan as Figure 6.28 shows for the case of Equation 6.12. When $p_m \geq 0.01$ and p_c is decreased (i.e., the schema theorem trade-off is violated) the number of function evaluations increases dramatically.

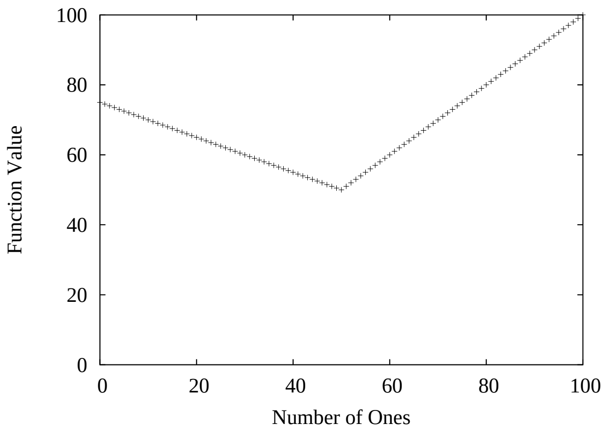


Figure 6.27: Fitness function as in Eq. 6.12

6.3.3 Open Paths in 8-Dimensional Hypercubes

We previously (Diaz-Gomez and Hougen, 2006d) suggested the fitness function

$$F(I) = \left(\frac{\sum_{j=0}^{2^d-1} (MP)_j - Penalty}{\sum_{j=0}^{2^d-1} (MP)_j} \right) \left(\frac{Length(S) + 1}{\#P} \right) \tag{6.13}$$

for finding open paths that obey the constraints just described, where *MP* is the result of the multiplication between the adjacency matrix, which encodes the hypercube, and the chromosome that is being evaluated; $Penalty$ is the number of genes in the chromosome that violate the constraints; i.e., number of nodes in the path without neighbors (Isolated points), number of nodes in the path with more than 2 neighbors (bad points), number of nodes in the path with one neighbor that exceed 2 (the head and the tail), number of genes with no neighbors (lazy points). $Length(S)$ is the length of the open connected path, and $\#P$ is the total number of nodes in the path.

Using fitness function as in Equation 6.13 in an eight dimensional hypercube, a snake of length 81 was obtained over 30 runs and an average snake length of 69.30 with a standard deviation of 6.70 (Diaz-Gomez and Hougen, 2006d). $1,000$ individuals were generated for the initial population with the first 128 genes filled with a 7-d snake of length 49. The last 128 bits were generated uniformly randomly turning off 50 bits in positions $p + 128$, where p is a position with allele value of 1 in positions 0 to 127, in order to preserve the embedded 7-d snake. The probability of one point crossover was

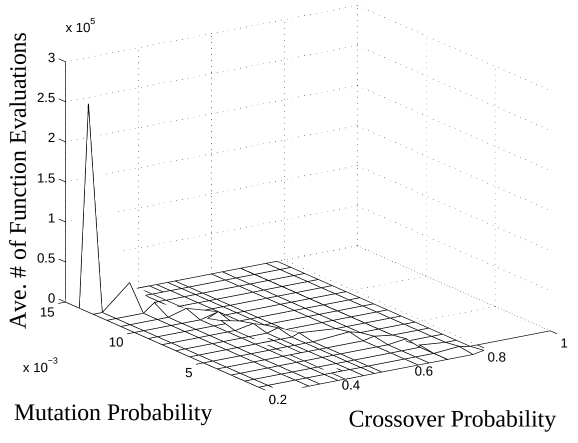


Figure 6.28: Average function evaluations to reach solution quality approx. $\mathbf{95.0}$.Fitness function as in Eq. 6.12.

60%, the probability of mutation was 2.4% per chromosome, crossover and mutation were performed above gene 128, and $1,000$ generations as the stop criterion was used (Diaz-Gomez and Hougen, 2006d).

As mutation and crossover are performed above gene 128, the schema under consideration corresponds to bits 128 to 255. The number of defining bits in those positions is 50—these were the bits turned off in order to maintain in the initial population the 7-d snake provided—and the distance $d(\xi) = 110$ because the first bit of the 7-d snake provided is in position 0 and the last is in position 110. With these parameters, and with $S_p = 1.5$ as previously (Diaz-Gomez and Hougen, 2006d), p_c should be ≤ 0.344 using Equation 5.30 with $p_m = 0.001$ per bit. An empirical study was performed to review this threshold, using $p_m = 0.001$ per bit (fixed), and $0 \leq p_c \leq 1.0$ per chromosome with 30 runs for each p_c and a step size 0.1. An 8-d snake of length 85 was found when $p_c = 30\%$, showing for this test set that it was better to use a small p_c in order to find a higher local maximum—see Table 6.23. This result outperforms previous findings of Bitterman (2004) using a GA, where it was obtained an average length of 75.5, the longest one of 76 with a standard deviation of 0.71.

In order to corroborate this finding, we performed tests again but this time the initial population was totally filled with zeros, i.e., the algorithm has to build the 8-d snake from scratch. The parameters used were population size 500 initially emptied, $2,000$

	$1,000$ *iterations,* $1,000$ *pop. size,* 30 *runs,* $p_m = 0.001$										
p_c	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$Max.$	82	83	83	85	82	82	80	82	83	84	82
$Aver.$	75.8	75.0	76.1	75.1	75.8	76.1	75.8	76.7	74.7	73.5	75.5
Std	3.43	3.90	4.07	4.49	3.49	2.93	2.66	2.41	4.04	6.28	3.58

Table 6.23: Finding snakes in a **8**-d hypercube. Initial population provided with one **7**-d snake of length **49**.

	$2,000$ *iterations,* 500 *pop. size,* 30 *runs,* $p_m = 0.001$										
p_c	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$Max.$	70	67	69	68	65	66	65	66	66	64	64
$Aver.$	59.3	60.2	58.7	58.9	59.0	57.5	56.9	58.4	56.6	57.7	58.1
Std	4.70	3.66	4.40	4.55	3.37	4.78	4.66	4.26	4.44	4.82	3.82

Table 6.24: Finding snakes in a **8**-d hypercube. Initial population from scratch.

generations, $p_m = 0.001$ per bit, and $0 \leq p_c \leq 1.0$ with step size 0.1. The longest 8-d snake found was 70 when $p_c = 0$, and the longer ones were found when $p_c \leq 30\%$, over 30 runs, for each set of parameters. See Table 6.24. This result outperforms Bitterman (2004) tests, over 30 runs, with GAs: (1) using random initial population, it was obtained an average length of 56.1 with a longest one of length 61 ($stdv = 3.76$), (2) using guided random initial population[11] it was obtained an average length of 57.70 with the best snake found with length 67 ($stdv = 4.27$), (3) using 1-point crossover it was obtained an average length of 58.6 with best snake found length of 69 ($stdv = 3.96$), (4) using mutation rate of 0.01 it was obtained an average of 59.4 with longest snake found of length of 63 ($stdv = 1.84$).

6.3.4 Selection Pressure

When the schema under consideration is small, the parameter that could have a high effect on the evolution of structure *A* is mutation. Section 5.2.1 shows that $p_m \rightarrow (S_p - 1)/S_p h$, so, the higher S_p is, the higher p_m should be. Theoretically, if $h \approx 50$, and if $S_p = 2$, $S_p = 4$, $S_p = 8$, $S_p = 16$, $S_p = 32$ and $S_p = 64$, then, the corresponding probability of mutation is $p_{2m} = 0.010$, $p_{4m} = 0.015$, $p_{8m} = 0.0175$, $p_{16m} = 0.01875$, $p_{32m} = 0.0194$, and $p_{64m} = 0.0196$, i.e., if $S_p \rightarrow 100$ then $p_m \rightarrow 1/h = 0.020$. Empirical studies were

[11]Term not defined in Bitterman (2004).

procedure τ			*Average*			*STDV.*		
S_p	p_c	p_m	*Qual.*	*Func. Ev.*	*# Mut.*	*Qual.*	*Func. Ev.*	*# Mut.*
2	1	0.000	99.7	2,521.9	0	1.40	222.9	0.0
4	1	0.000	96.0	1,399.0	0	6.73	337.6	0.0
8	1	0.000	91.5	925.2	0	8.42	353.6	0.0
16	1	0.000	79.6	469.7	0	9.75	276.3	0.0
32	1	0.000	72.6	273.3	0	4.31	78.4	0.0
64	1	0.000	67.0	173.3	0	4.63	52.0	0.0
2	1	0.010	100.0	3,885.4	3,930.2	0.00	308.7	320.9
4	1	0.015	100.0	2,210.4	3,387.8	0.00	232.3	356.4
8	1	0.017	97.3	1,567.7	2,908.2	7.49	404.4	761.3
16	1	0.019	93.5	1,160.6	2,253.9	7.81	476.5	968.6
32	1	0.019	80.36	498.4	952.7	7.93	298.5	572.8
64	1	0.020	72.9	276.6	556.9	5.27	89.76	189.9

Table 6.25: Influence of selection pressure on the quality of the solution for the One-Max Function. First part with no mutation. Second part, when Schema Theorem trade-off is obeyed. $p_c = 1$.

performed with the optimal GA described in Harik and Lobo (1999), however, changing the selection pressure S_p from 2 to 64 in steps of power of 2, over 30 runs for each S_p, diminishes the quality of the solution, as well as the number of function evaluations (see top part of Table 6.25). But, if the mutation probability p_m is changed according to S_p, then, there is an improvement on the quality of the solution, as is shown in Table 6.25 (bottom part) and Figure 6.29.

In order to continue with the relationship between selection pressure and mutation probability, when the schema under consideration is long, as in the case of snakes in the 8-dimensional hypercube ($\epsilon = 110/127 = 0.866$), test for the cases of $p_c = 0.3$ and $p_c = 0.7$, which were the best results obtain as is shown in Table 6.23, are shown in Table 6.26.[12] If $p_c = 0.3$, $p_m = 0.001$ and S_p is increased, then, there is no improvement in the average quality of the solution with respect to $S_p = 1.5$, as is shown in the top left of Table 6.26. As a first case, if the mutation probability is increased according to the schema theorem trade-off (Equation 5.31), then, there is an improvement for the cases of selection pressures above 4 comparing with previous case (See top right of Table 6.26). Now, as a second case, if the crossover probability is increased to 0.7 (maintaining $p_m = 0.001$ fixed), then, there is not a special tendency in the improvement of the average quality of

[12]The case where $p_m = 0.001$ is not base on Equation 5.31. Those entries were used for comparison purposes.

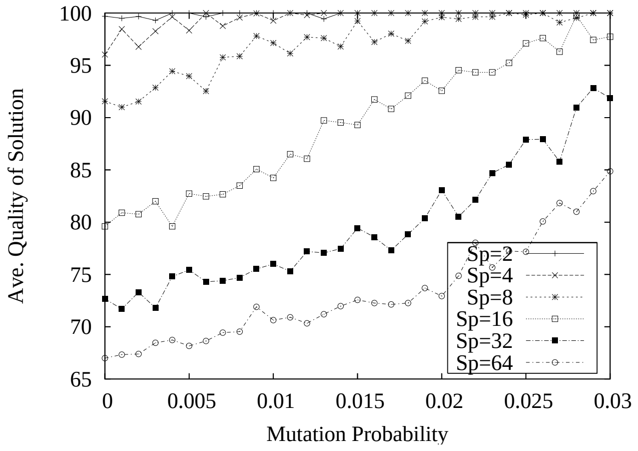


Figure 6.29: Average quality of the solution for different S_p and $0 \leq p_m \leq 0.03$. As S_p is increased, p_m should be increased in order to obtain better quality. One-Max Function. **30** runs per set of parameters.

the solution (except that all average values are lower for $S_p > 1.5$ and that a higher snake appears when $S_p = 8$, see bottom left of Table 6.26), but, if besides the increasing of the crossover probability to 0.7, there is an increase in the mutation probability in accordance to p_c and S_p, then, there is an increase in the average quality of the solution above $S_p = 2$, comparing top left with bottom right of Table 6.26.

In order to compare results, as in Table 6.23, changing *only* selection pressure S_p, tests were performed again, increasing S_p, as is shown in Table 6.27. With the parameter setting established: $p_m = 0.001$ fixed and, S_p and p_c varying—but not in accordance with the schema theorem trade-off, on average (over all p_c) the quality of the solution diminished (see Figure 6.30 where top line corresponds to $S_p = 1.5$ and bottom line corresponds to $S_p = 64$).

Proportional Selection Proportional selection is perhaps the one that is closest to the schema theorem: $S_p = \hat{\mu}_\xi(t)/\overline{\mu}(t)$; however, in tests performed presently, the pressure of proportional selection was not enough to climb to "good" solutions. For the case of the one-max function, using the 100 individuals, $p_c = 1.0$, proportional selection, an adapting mutation probability as $p_m = (S_p - 1)/S_p\overline{\mu}(t)$, where $S_p = Max(fitness(t))/\overline{\mu}(t)$, i.e., $\hat{\mu}_\xi(t) \approx Max(fitness(t))$ and $h \approx \overline{\mu}(t)$, and the algorithm stops when it reaches the global maximum or when a column is totally zeroed. Figure 6.31 shows the maximum

S_p	p_c	p_m	*Max.*	*Ave.Q.*	*STDV.*	p_m	*Max.*	*Ave.Q.*	*STDV.*
1.5	0.3	0.001	85	75.1	4.49	0.001	85	75.1	4.49
2	0.3	0.001	81	73.3	4.47	0.006	72	67.3	2.24
4	0.3	0.001	83	74.1	4.04	0.013	73	68.4	1.94
8	0.3	0.001	81	73.3	4.67	0.016	79	75.1	2.93
16	0.3	0.001	79	73.9	4.54	0.018	84	76.7	4.17
32	0.3	0.001	81	72.7	3.65	0.019	85	77.3	4.68
64	0.3	0.001	82	71.6	5.34	0.019	85	77.8	4.75
1.5	0.7	0.001	82	76.7	2.41	0.001	82	76.7	2.41
2	0.7	0.001	81	73.7	4.44	-0.005	-	-	-
4	0.7	0.001	80	73.7	3.67	0.007	81	75.8	4.20
8	0.7	0.001	86	74.1	5.24	0.013	85	77.8	3.08
16	0.7	0.001	85	73.0	5.82	0.016	84	76.8	3.47
32	0.7	0.001	83	72.2	5.10	0.018	83	76.4	3.79
64	0.7	0.001	84	73.9	5.97	0.019	85	78.3	3.55

Table 6.26: Influence of selection pressure on the quality of the solution for snakes in 8-dimensional hypercubes. Left part change of S_p without mutation change. Right part, when Schema Theorem trade-off is obeyed. $1,000$ Iterations. $1,000$ individuals. 30 runs per (S_p,p_c,p_m).

quality of the solution, and Figure 6.32 shows the average of the solution quality with the standard deviation. The algorithm did not reach the global maximum for the test set specified. The average of the selection pressure was $\overline{S}_p = 1.20$—see Figure 6.33—and the average of the mutation probability was $\overline{p}_m = 0.00298$—see Figure 6.34—over 30 runs with an average of $3,476.67$ function evaluations per run.

For the case of open paths in hypercubes, "small" probabilities of crossover ($p_c = 0.2$) and mutation ($p_m = 0.001$) were chosen in order to compare results with Table 6.23. Figure 6.35 shows the proportional selection pressure, the average was $\overline{S}_p = 1.14$ over 30 runs ($30,000$ generations). Table 6.28—where $\#P$ is the number of points in the path, HT is the number of head-tail, $Is.$ is the number of isolated points and Bad is the number of bad points—shows the configuration of paths found; there were violations of the constraints in all of them, i.e., there were no snakes in each of the 30 runs.

6.3.5 Additional Contributions

Plan τ as presented in Section 5.1.3.1 in Table 5.2 uses as Ω parameters, a probability of crossover (p_c) of 60% and a probability of mutation per chromosome (p_m) of 3%. How

	$S_p = 1.5$										
p_c	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$Max.$	82	83	83	85	82	82	80	82	83	84	82
$Aver.$	75.8	75.0	76.1	75.1	75.8	76.1	75.8	76.7	74.7	73.5	75.5
Std	3.43	3.90	4.07	4.49	3.49	2.93	2.66	2.41	4.04	6.28	3.58
	$S_p = 2$										
$Max.$	82	83	81	81	81	80	80	81	83	82	80
$Aver.$	72.5	73.5	73.2	73.3	74.5	73.2	72.3	72.9	73.2	73.6	73.1
Std	6.27	5.18	5.43	4.47	3.86	4.63	4.79	3.48	5.01	3.60	4.81
	$S_p = 4$										
$Max.$	79	85	82	83	79	81	85	79	80	80	83
$Aver.$	72.6	74.1	74.3	74.1	73.3	72.4	73.7	72.1	72.4	72.5	74.0
Std	4.28	5.74	4.62	4.04	4.01	5.05	4.71	4.77	5.05	5.73	3.65
	$S_p = 8$										
$Max.$	78	83	83	81	81	81	78	79	83	81	81
$Aver.$	71.8	73.2	73.7	73.3	72.6	73.7	72.8	72.6	73.0	73.1	73.3
Std	3.99	5.88	4.60	4.67	4.61	4.12	4.90	4.63	5.29	4.43	4.74
	$S_p = 16$										
$Max.$	83	83	81	79	80	81	81	86	85	82	80
$Aver.$	72.6	74.4	72.5	73.9	74.5	72.4	71.3	72.5	73.5	73.4	72.9
Std	5.20	5.16	5.48	4.54	4.24	5.12	5.91	6.85	4.42	5.10	3.82
	$S_p = 32$										
$Max.$	85	80	79	81	84	78	79	83	85	82	81
$Aver.$	72.5	70.7	73.0	72.7	74.3	70.3	71.3	73.5	71.8	70.6	72.4
Std	5.42	5.35	4.79	3.65	4.74	5.21	4.98	4.49	5.01	6.69	5.32
	$S_p = 64$										
$Max.$	79	82	81	82	80	83	80	78	80	83	82
$Aver.$	72.4	71.6	74.0	71.6	72.6	73.4	72.2	70.7	72.6	72.0	73.5
Std	5.18	6.27	5.09	5.34	4.67	5.08	5.40	4.31	4.82	5.80	4.86

Table 6.27: Selection pressure and its influence on the quality of the solution for snakes in 8-dimensional hypercubes. Initial population provided with one **7**-dimensional snake of length **49**. $p_m = 0.001$. $1,000$ Iterations. $1,000$ individuals. 30 runs per each parameter set.

#P	80	81	81	81	79	78	82	78	85	82	81	78	80	85	80	84	79	81	77	81	81	81	81	81	78	82	81	78	82	76
Len.	71	75	65	75	72	65	76	73	76	76	71	70	57	74	76	82	75	70	69	72	77	79	71	71	63	76	57	57	68	67
HT	2	5	7	4	2	6	4	2	6	2	6	6	14	4	2	2	2	8	2	2	2	2	4	6	8	4	12	8	8	4
Is.	3	1	3	10	7	8	3	1	9	9	14	5	23	20	13	8	8	5	15	5	6	12	5	4	8	5	9	7	22	10
Bad	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	4	0	0	0	4	0	0	0

Table 6.28: Characteristics of open paths obtained when proportional selection used. Initial population provided with one **7**-dimensional snake of length **49**. $p_c = 0.2$, $p_m = 0.001$. $1,000$ Iterations. $1,000$ individuals. 30 runs.

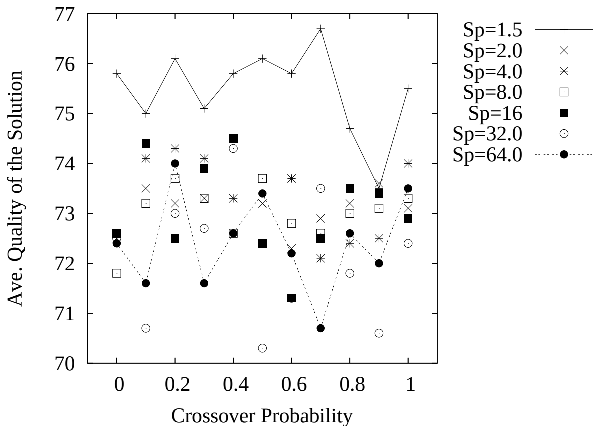


Figure 6.30: Selection pressure and its influence on the average quality of the solution for snakes in 8-dimensional hypercubes. Initial population provided with one $\mathbf{7}$-dimensional snake of length $\mathbf{49}$. $p_m = 0.001$. $1,000$ Iterations. $1,000$ individuals. 30 runs per each parameter set.

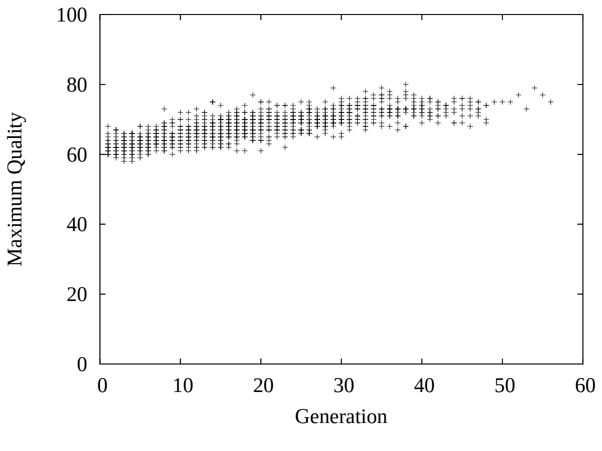


Figure 6.31: One-max function; Maximum quality of the solution per generation when proportional selection used; $\boldsymbol{\overline{S}_p = 1.20}$, $\boldsymbol{p_c = 1.0}$ $\boldsymbol{\overline{p}_m = 0.00298}$; $\mathbf{30}$ runs.

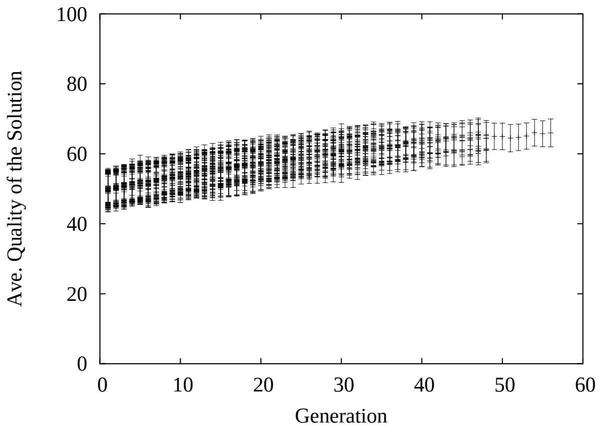


Figure 6.32: One-max function; Average quality of the solution per generation when proportional selection used; $\overline{S}_p = 1.20$, $p_c = 1.0$ $\overline{p}_m = 0.00298$; 30 runs.

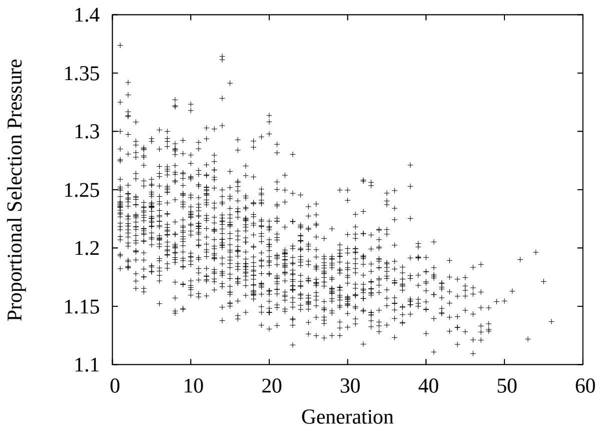


Figure 6.33: One-max function; Proportional selection pressure per generation over 30 runs.

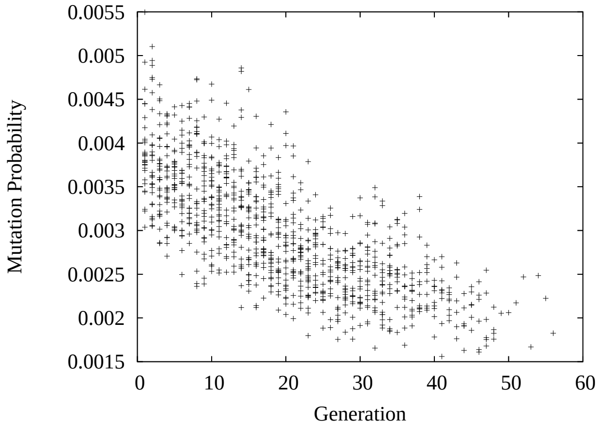


Figure 6.34: One-max function; Mutation probability per generation when proportional selection used; $\overline{S_p} = 1.20$; 30 runs.

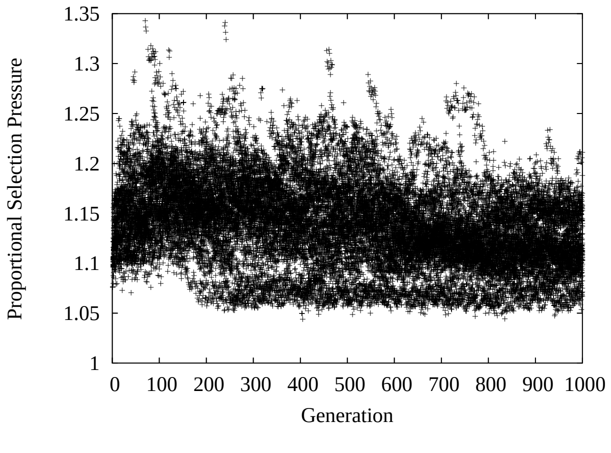


Figure 6.35: Open paths in 8d-hypercubes; Proportional selection pressure per generation over 30 runs.

	$p_m = 3\%$				p_m as in Eq. 6.14			
Fitness Function	*Minimum*	*Maximum*	*Average*	σ	*Minimum*	*Maximum*	*Average*	σ
Eq. 6.5	26	$17,032$	$2,874.2$	$4,104.0$	1	42	13.3	11.1
Eq. 6.6	6	$25,352$	$3,174.1$	$5,414.7$	1	92	25.8	23.2
Eq. 6.7	8	$20,435$	$2,044.4$	$4,187.6$	2	85	26.7	19.0
Eq. 6.8	2	$9,807$	$2,739.6$	$3,032.2$	4	86	27.4	22.4
Eq 6.9	4	$15,138$	$2,770.3$	$3,939.4$	2	103	30.1	24.1
Eq. 6.10	1	$13,089$	$1,750.5$	$3,015.5$	3	154	30.3	29.5
Eq. 6.11	5	$17,117$	$2,295.5$	$4,206.4$	3	82	21.5	21.2

Table 6.29: Results for hunting longest snakes in a 4-dimensional hypercube; different fitness functions; mutation rate $p_m = 3\%$ and as in Equation 6.14; 30 runs.

can plan τ be improved so that the number of generations and the standard deviation are reduced?

For the 4-dimensional hypercube, it turns out that a parameter $\omega_t \in \Omega$ capable of producing a large change is mutation because with the population of size 10, once the algorithm converges, if it has not reached an optimum, then in order to obtain it, the operator that could still make it is mutation. The mutation probability was changed so that it is applied in accordance with the average fitness value of the entire population.

Plan τ_m is such that the mutation parameter is variable according to

$$p_m(t+1) = \frac{\mu_{max}(t) - \overline{\mu}(t)}{\mu_{max}(t)} * 100\% \tag{6.14}$$

where $\overline{\mu}(t)$ is the average fitness value of the population at step t (generation t).

Equation 6.14 is such that if at time step t the normalized average fitness value of the population is quite far from the optimum of 1, then $p_m(t+1)$ is high. If, on the contrary, the average normalized fitness value of the entire population is quite near the optimum, then $p_m(t+1)$ is going to be low. Comparative results with a fixed probability of mutation per chromosome of 3%, are shown in Table 6.29. Table 6.30 shows the corresponding results when Equation 6.14 is used. Equation 5.17 is statistically significantly different from all the others except Equation 5.23.[13]

Table 6.31 compares the average number of generations to find a maximum when Equation 6.14 is used against other mutation rates. A fixed mutation rate of 3%, as illustrated previously; mutation rates of 100% after generation $1,000$, 500, and 250; and a

[13]Here, we are facing a multiple comparison problem, however, this is the exploratory study, where we are testing different parameters, in order to see the possibility to use some of them in the test of hypothesis.

	p_m *as in Equation* 6.14					
Eq.	6.5	6.6	6.7	6.8	6.9	6.10
Eq. 6.6	0.010	-	-	-	-	-
Eq. 6.7	0.001	0.870	-	-	-	-
Eq. 6.8	0.003	0.791	0.901	-	-	-
Eq 6.9	0.001	0.491	0.554	0.658	-	-
Eq. 6.10	0.005	0.517	0.580	0.669	0.973	-
Eq. 6.11	0.065	0.457	0.322	0.302	0.151	0.192

Table 6.30: T-test for hunting longest snakes in a 4-dimensional hypercube; different fitness functions; adaptive mutation rate; number of generations until longest snake found; Probability of results given the null hypothesis is shown; 30 runs per fitness function

	Mutation rate						
Equation	Fixed 3%	S1000	S500	S250	Generation	Pop*Generation	(1-$\overline{\mu}(t)$)*100
Eq. 6.5	2,874.2	744.3	394.4	194.5	189.9	62.5	13.3
Eq. 6.6	3,174.1	670.7	381.0	211.4	141.8	42.0	25.8
Eq. 6.7	2,970.8	607.5	322.2	181.0	164.0	34.0	26.7
Eq. 6.8	3,356.4	500.5	382.9	215.7	172.2	46.5	27.4
Eq 6.9	1,346.4	551.6	329.1	237.9	167.8	40.6	30.1
Eq. 6.10	1,451.7	696.1	379.1	235	139.9	39.7	30.3
Eq. 6.11	3,264.2	411.7	373.2	202.5	139.2	50.4	21.5
Totals							
	17,648.6	4,182.4	2,561.9	1,478	1,114.8	315.7	175.1

Table 6.31: Average number of generations for hunting longest snakes in a 4-dimensional hypercube; different fitness functions; different mutation rates; 30 runs.

mutation rate equal to the generation, a mutation rate equal to the number of individuals in the population (10) multiplied by the generation are used.[14] As the mutation rate increases, the number of generations to reach the maximum diminishes.[15] Some mutation rates begin fixed and get a full rate like *S1000*, others increase gradually like *Generation*, but Equation 6.14 begins high and decreases. We obtained the idea of Equation 6.14 from observation, in the sense that we wanted to diminish the average number of generations and "good" results were obtained by increasing the mutation rate.[16]

If a comparison is made between different mutation rates using a fitness function as in Equation 6.7, histograms from Figures 6.36 to 6.40 show the net effect of mutation

[14]It should be taken into account that the mutation rate is per chromosome, i.e., a mutation rate of 100% per chromosome of length $l = 16$ means an effective mutation of 6.25% per bit.

[15]Possibly the cause of this factor is that the length of the chromosome is short (16 bits).

[16]The proposal and testing of mutation rates occured in the order specified in Table 6.31.

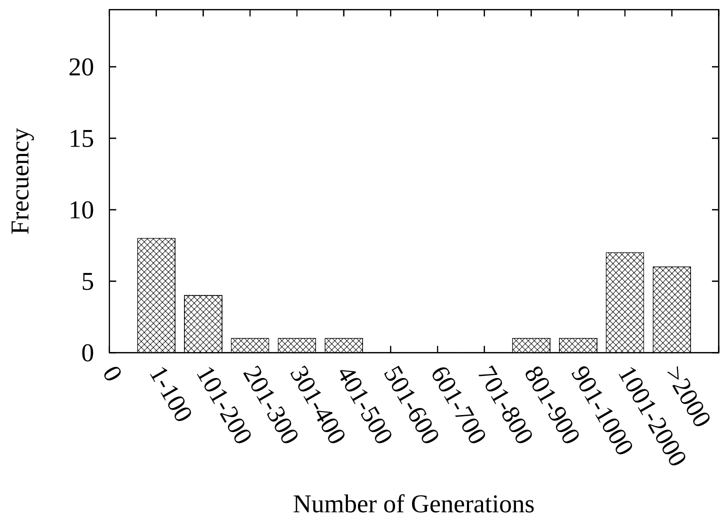


Figure 6.36: Histogram for longest snakes using Equation 6.7; generations until global maximum reached; mutation rate 3% fixed in all generations.

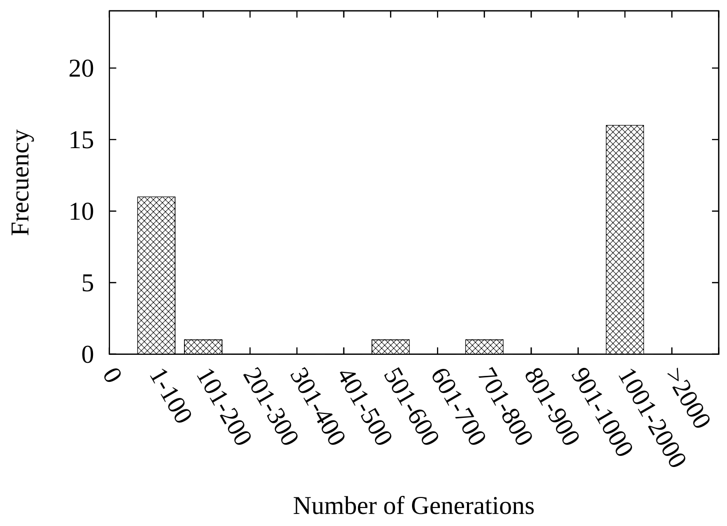


Figure 6.37: Histogram for longest snakes using Equation 6.7; generations until global maximum reached; mutation rate 3% when generation $< 1,000$ and 100% when generation $\geq 1,000$.

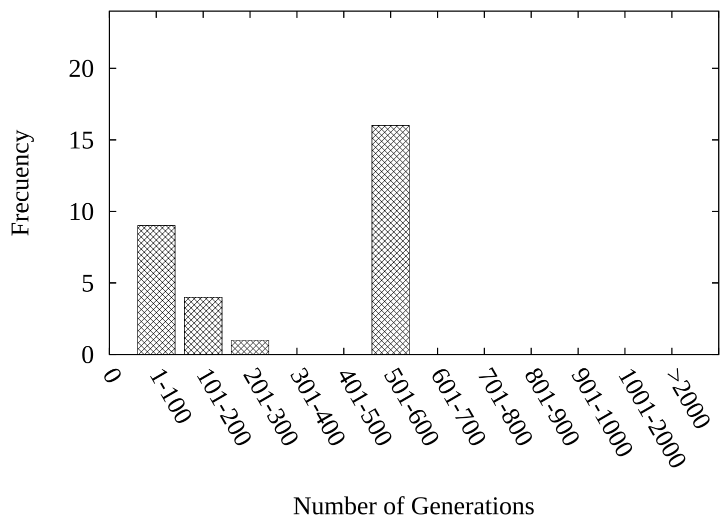


Figure 6.38: Histogram for longest snakes using Equation 6.7; generations until global maximum reached; mutation rate 3% when generation < 500 and 100% when generation ≥ 500.

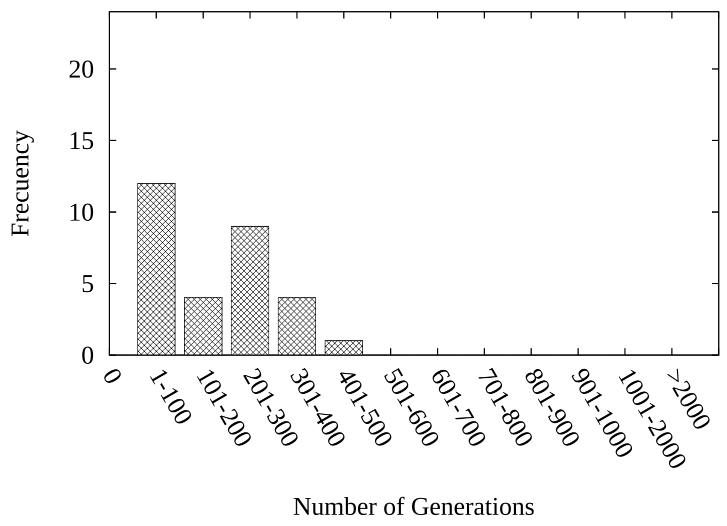


Figure 6.39: Histogram for longest snakes using Equation 6.7; generations until global maximum reached; mutation rate 3% when generation < 250 and 100% when generation ≥ 250.

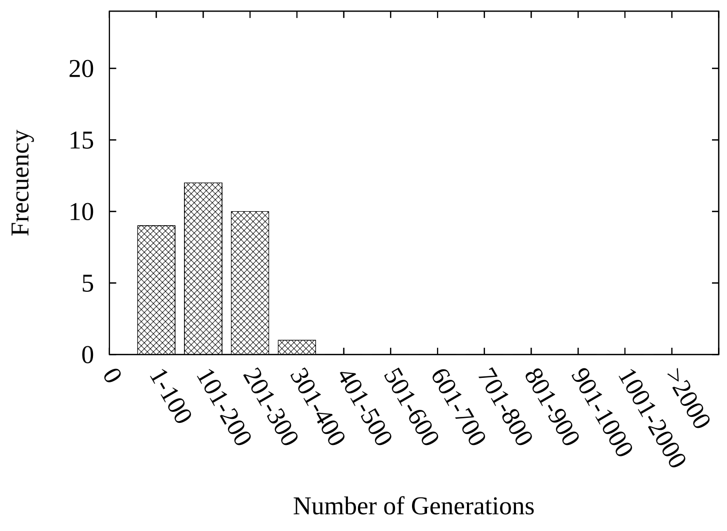


Figure 6.40: Histogram for longest snakes using Equation 6.7; generations until global maximum reached; mutation rate percentage equal to the generation.

rate on the number of generations with each mutation rate. For snakes in hypercubes of dimension 4, mutation rate is like the “blade” that cuts the number of generations down when it is increased. When the mutation rate is constant and equal to 3% in all generations, the average number of generations to find a longest snake is $2,044.4$ with a maximum of $20,435$ generations; however, if the mutation rate is changed to 100% after generation $1,000$, the average is 607.5 with a maximum of $1,078$ generations; if the mutation rate is changed to 100% after generation 500, then the average number of generation to find a longest snake is 322.2 with a maximum of 564 generations; and if the mutation rate is changed to 100% after generation 250, the average number of generation is 181.0 with a maximum of 412 generations. The bimodality presented in previous cases is changed to an unimodal distribution when the mutation rate is variable according to the number of generations—see Figure 6.40.[17]

6.3.6 Conclusions

Section $6.3.1$ presents the performance of the crossover and mutation operators with respect to the quality of the solution for the one-max function, with and without selection

[17]When the mutation rate is variable and changed to $Pop * Generation$ or $(1 - \overline{\mu}(t)) * 100$ the number of generations is less than 100, so there is no distribution according to the range selected.

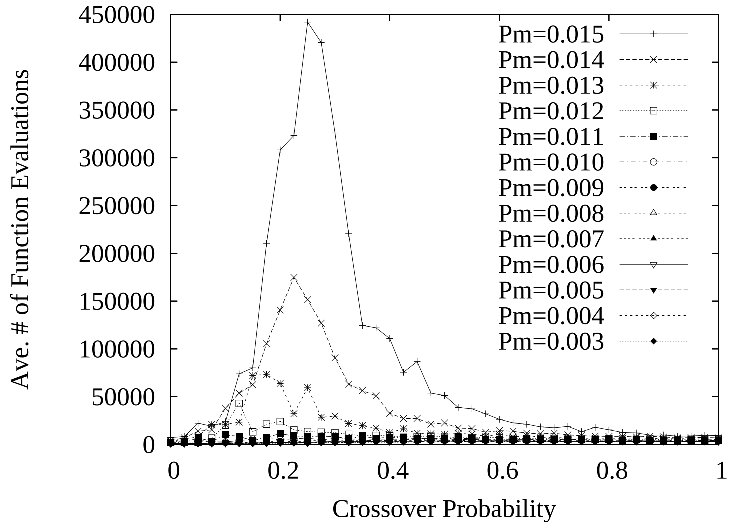


Figure 6.41: Average function evaluations to reach solution quality as in Fig. 6.25; $\boldsymbol{0.0 \leq p_m \leq 0.015}$, step size $\boldsymbol{0.001}$.

pressure. But quality has a cost. Figure 6.41 shows the average number of function evaluations performed by the GA for different procedures. Table 6.21 shows the average number of function evaluations and mutations performed with a mutation probability of 0.015 and also without mutation. The impact on the average number of function evaluations when a probability of mutation of 0.015 (greater than 0.01, violating the schema theorem trade-off) is used is evident. Another way to look at the problem with this cost is that when p_m is increased and p_c is decreased, (i.e., when the trade-off of the schema theorem is not followed for this type of problem) the cost is going to be high. From Figure 6.25, it can be inferred that quality can be reached by increasing the probability of mutation and decreasing the probability of crossover; but, as just stated, the number of function evaluations is going to increase—see Section 6.3.2 for the one-max variant.

Figure 6.26 shows the net effect of procedures that use selection versus no selection at all, and Table 6.32 shows the result of those procedures and the corresponding costs on the averages of function evaluations for various p_m. It can be seen the big difference between $p_m \leq 0.009$ and $p_m \geq 0.01$, i.e., when there is no violation of the schema theorem trade-off, and when there is: if $p_c \to 1$, and $l >> S_p$, then $p_m \to (S_p - 1)/S_p h$ (see Section 5.2.1) and, as $l = 100$, $S_p = 2$, and $h \approx 50$ (since it is expected that there will be approximately 50 ones and 50 zeros in each chromosome), then $p_m \to 0.01$.

procedure τ			*Qual.*		*Func. Ev.*	
S_p	p_c	p_m	*Ave.*	*std.*	*Ave.*	*std.*
1	0	0.000	56.86	2.72	3,156.66	936.12
1	0	0.009	58.70	2.84	8,360.00	5,388.21
1	0	0.015	59.83	3.33	53,940.00	50,916.77
1	1	0.000	60.56	2.26	2,336.66	569.62
1	1	0.009	60.10	3.19	6,766.66	3,454.36
1	1	0.015	60.70	1.82	50,956.66	34,626.19
2	0	0.000	61.86	2.01	650.00	73.10
2	0	0.009	70.16	5.50	1,726.66	2,137.01
2	0	0.015	79.46	7.54	7,210.00	9,276.45
2	1	0.000	99.70	1.46	2,521.90	222.97
2	1	0.009	100.00	0.00	3,665.30	365.75
2	1	0.015	100.00	0.00	9,305.73	4,037.58

Table 6.32: Average solution quality and cost for different plans for the one-max function.

Similar results were obtained using Equation 5.31. This is corroborated in Figures 6.28 and 6.41, where it can be appreciated how, for $p_m \geq 0.01$, the average number of function evaluations grows. This tendency occurs in the one-max function and its variant.

If having no selection pressure makes little progress in the quality of the solution, as shown in Table 6.22, having a quite high selection pressure makes the algorithm, for the one-max function, converge quickly with less quality of the solution as is shown in Table 6.25. However, if the Schema Theorem Trade-Off is obeyed—i.e., $p_m \rightarrow (S_p - 1)/S_p h$—an improvement of the quality of the solution is obtained, as is shown in Table 6.25 and Figure 6.29, with no guarantee to reach the 100%. But, if the Schema Theorem Trade-Off is violated for this problem, i.e., for $p_c = 1.0$, if $p_m > 1/h = 0.02$ when S_p is high, then the number of function evaluations is going to be prohibited to get convergence with no guarantee to reach the global optimum either (see Figure 6.42 for the average of function evaluations when S_p is increased and $p_m > 0.02$).

It is a difficult problem finding snakes in the 8-dimensional hypercube, not only because the search space is large (2^{2^8} encoding the path as a 2^8-array of ones and zeros) but also because of the constraints imposed—every time a point is added, it restricts the use of other points in an asymmetric way. The genetic operators crossover and mutation seem to be quite destructive in this case. The tests performed, as defined in Section 6.3.3, showed that p_c and p_m should be maintained "small". So, if p_c and p_m are increased, then, the number of generations are going to increase and/or the quality of the solution

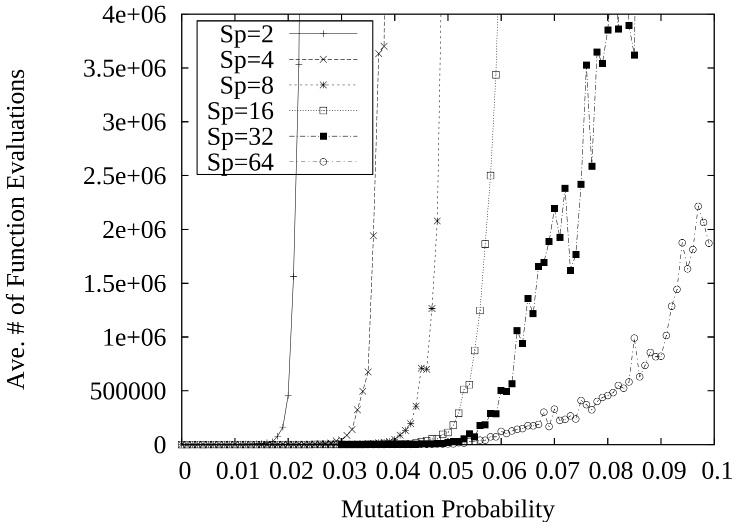


Figure 6.42: Average of function evaluations for $p_c = 1.0$, different p_m and S_p increased. If $p_m > 1/h = 0.02$ then Ave. # of iterations is prohibited. One-Max Function. **30** runs per set of parameters.

is going to decrease. The probability of crossover and the probability of mutation were varied to corroborate the schema theorem trade-off for this particular problem. The parameters used were a maximum of $2,000$ generations, a population size of 500 initially zeroed, $0.0012 \leq p_m \leq 0.002$ per bit with step size 0.0002, $0 \leq p_c \leq 100\%$ for one point crossover with step size 10%, and $S_p = 1.5$. While the algorithm runs, it stores snakes it finds by looking for snakes with fitness value ≥ 0.95. If it finds a snake with fitness value ≥ 0.95, it stops; otherwise it continues until generation $2,000$, reporting the findings. If a snake is not found the algorithm diverges for the parameters established. Figure 6.43 shows how the average number of function evaluations increases (lines above 1000 in the y axes) and how the number of times (amplified by 25) the algorithm reached a quality of solution $\geq 95\%$ decreases, for this test set, when p_m and p_c increase, i.e., when the schema theorem trade-off is violated. Each line corresponds to a constant p_m. For mutation rates 0.018 and 0.02, the algorithm never reaches the quality stipulated in the $2,000$ generations.

The test performed at the end of Section 6.3.4 for the case of the snake problem showed a slight diminishing of the quality of the solution when the only parameter that changes is

	# times $\geq 95\%$		*# of Generations*	
S_p	*Ave. of ave.*	*STDV*	*Ave. of ave.*	*STDV*
1.5	24.8	2.52	1,069.2	138.77
2	28.0	1.14	466.2	77.70
4	28.4	1.21	378.8	63.87
8	29.0	0.54	309.9	59.15
16	28.8	1.17	367.2	81.77
32	28.5	1.04	370.4	86.91
64	27.4	1.86	441.5	114.46

Table 6.33: Ave. of ave. of # of times the solution quality of $\geq 95\%$ is reached and, ave. of ave. of # of generations, over all p_c as in Figure 6.44, when S_p is increased. Initial population from scratch. $\boldsymbol{p_m = 0.0012}$ constant. Max. generations $\mathbf{2,000}$. $\mathbf{30}$ runs.

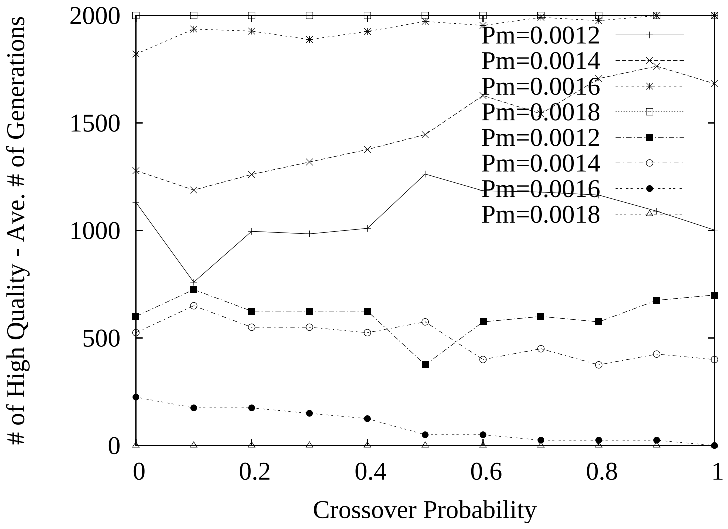


Figure 6.43: Ave. generations to reach solution quality $\boldsymbol{\geq 95\%}$ (top lines above 1000). Number of times algorithm reaches quality $\boldsymbol{\geq 95\%}$ (bottom lines under 1000). $\boldsymbol{0.0012 \leq p_m \leq 0.002}$, step size $\mathbf{0.0002}$. Max. generations $\mathbf{2,000}$. $\mathbf{30}$ runs.

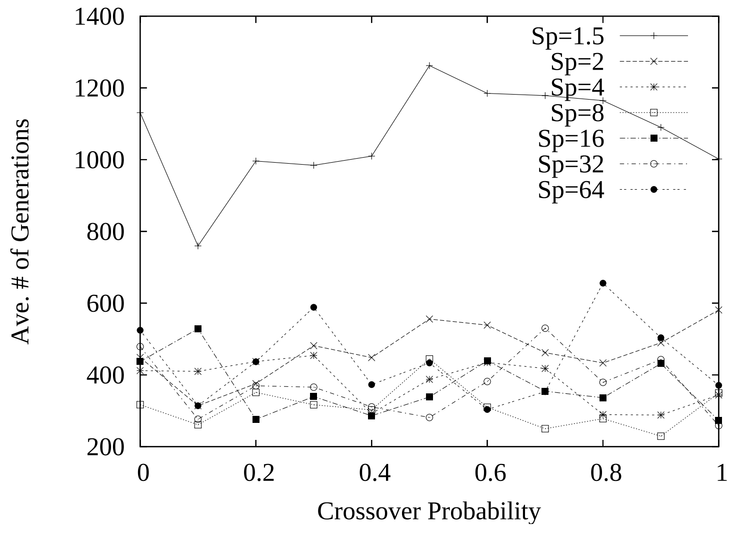


Figure 6.44: Ave. # of generations to reach a solution quality of $\geq \mathbf{95\%}$ when S_p is increased. Initial population from scratch. $\boldsymbol{p_m = 0.0012}$ constant. Max. generations $\mathbf{2,000}$. $\mathbf{30}$ runs.

selection pressure S_p (see Table 6.27 and Figure 6.30). Tests were performed again, initiating the algorithm with the population totally zeroed—500 individuals—$p_m = 0.0012$, $0 \leq p_c \leq 1$ and increasing S_p in order to reach a quality $\geq 95\%$. As Figure 6.44 shows, the number of generations diminished with respect to $S_p = 1.5$. Table 6.33 shows the average of the average (over p_c values tested) of the number of times the algorithm reaches a quality $\geq 95\%$ and the average of the average of the number of generations over all p_c. For this test set, the average of the average of the solution quality increases, and the average of the average of the number of generations begins to decrease for $S_p \leq 8$ (see Table 6.33). It is possible that changing the initial condition to initiate the algorithm with an initial population totally zeroed helps to reach the quality $\geq 95\%$ in less generations when $1.5 \leq S_p \leq 8$.

For both problems, with the test set defined and results shown in Section 6.3.4, there was no improvement in the solution quality when proportional selection is used. It seems that the selection pressure was not high enough to climb to better solutions (the average of the proportional selection pressure was less than 1.2 in both cases, see Figures 6.33 and 6.35). Besides the empirical studies in regarding the schema theorem trade-off, additional contribution was done with the empirical study about the impact of the mutation rate in finding the global optimum for snakes in 4-dimensional hypercubes. Mutation rate according to Equation 6.14 helps the algorithm to find higher quality on the solution (7 length snakes) with fewer function evaluations (see Table 6.31).

Chapter 7

Hypotheses, Experiments, and Results

We have concluded the exploratory study that allows us to understand the research questions more clearly, so now we can formulate specific testable hypotheses, run experiments, and present results. This chapter, then tests the theory developed in Chapter 5 around the three research statements proposed in this dissertation (see Section 1.1): Is there a correlation between diversity in the initial population and the performance of the algorithm, measured in the quality of the solutions found and/or the number of generations to find "good" solutions? do internal parameters combined with the union operator result in better performance than external parameters (i.e., exogenous parameters that may be tuned according to the input)? and, is there a relationship between selection pressure and the crossover and mutation probabilities that could help GAs in finding better solutions? Besides the two problems analyzed in Sections 3.1 and 4.1, the classic problem, the *one-max problem*, which is addressed extensively in the GA literature (Cheng and Kosorukoff, 2004; Harik and Lobo, 1999; Giguere and Goldberg, 1998; Yu et al., 2003), has been used for comparison purposes. These problems have been chosen because they represent problems where the schema under consideration is small ($d(\xi) = 1/(l-1)$ for the one-max problem and the intrusion detection problem) and problems where the schema under consideration is long ($d(\xi) \approx 1$ for the snake-in-the-box problem). Considering that these two kind of problems make results more general and possibly of more applicability.

The three general research questions that were stated in Chapter 1 are here refined into specific, testable hypotheses, which are presented along with appropriate experiments and results, in the same order as in Chapter 1.

7.1 Diversity & Performance

It is widely believed that if the initial population is more diverse, then the performance of the algorithm may be improved (see for example Jaroslaw Arabas and Mulawka (1995); Bitterman (2004); Burke et al. (2004); Lobo and Lima (2005); Grefenstette (1986); McPhee and Hopper (1999) and Rosca (1995)). Our purpose with this research, then, is to do an empirical study regarding the relationship between diversity in an initial population and GA performance. Do initial populations that are more diverse give better performance than populations with lower diversity? If this is the case, then it would be beneficial to look for initial populations where diversity value is higher. However, if this is not the case, then we need to look for factors of initial populations that do impact the performance; one such factor could be population size. It is also possible that a single factor alone may not be.

Research Question 1: The structure $A \in \mathcal{A}$: Is there a correlation between diversity in the initial population and the performance of the algorithm, measured in the quality of the solutions found and/or the number of generations to find "good" solutions?

General Form of Hypothesis 1.

$$\text{If } V(P_A) \geq V(P_B) \text{ then } X(G, P_A) \geq X(G, P_B) \tag{7.1}$$

where $V(P)$ is the diversity of population P and $X(G, P)$ is the expected performance of a genetic algorithm G with population P. Expected performance is measured as the expected solution quality of the best solution found so far after a given number of generations or the expected number of function evaluations to obtain a solution of a given quality. Diversity is measured as the difference between genes according to the entropy metric and as the difference between individuals according to the Hamming metric in terms defined in Section 5.0.2.

Two problems are going to be tested, one with small schema (the one-max problem) and one with long schema (the snake-in-the-box problem). The performance of the GA is going to be measured as the quality of the solution found so far in a fixed number of generations, and as the number of generations to reach the global maximum. These give a set of eight specific hypotheses to test for the general hypothesis 1, as can be seen in Table 7.1. As we are going to test eight different hypotheses at the 95% confidence level

	One-Max				*Snake-in-the-box*			
	Diversity Metric				*Diversity Metric*			
	Entropy		*Hamming*		*Entropy*		*Hamming*	
	Performance Metric							
Hypothesis	*Quality*	*Generations*	*Quality*	*Generations*	*Quality*	*Generations*	*Quality*	*Generations*
1.1	X							
1.2		X						
1.3			X					
1.4				X				
1.5					X			
1.6						X		
1.7							X	
1.8								X

Table 7.1: Specific hypotheses to test for general hypothesis 1.

or greater, in three trials, there is possibility that we may obtain a false positive result (one that could appear statistically significant but is not). To avoid this multiple comparison problem, if we find one result that appears statistically significant for a particular combination of metric and outcome, we will run the test set a second time for that particular combination of metric and outcome, generating a new set of results to look for a truly significant result.

To form specific, testable hypotheses, we need to specify a set of other parameters for the GA used in testing.

Parameter set 1.1: Population size $N = 20$, chromosome length $l = 64$, 2-tournament selection, probability of uniform crossover $p_c = 1.0$, probability of mutation $p_m = 0.001$ per bit, stop criterion is a maximum of $100,000$ generations.

Parameter Justification: As Lobo and Goldberg (2004) suggested for a 64 bit length chromosome 64 individuals, we used 20 individuals in order to have a more diversity differentiation and possible to avoid a ceiling effect. Besides that, Lobo and Goldberg (2004) used 2-tournament selection, a probability of uniform crossover of $p_c = 1.0$, a probability of mutation $p_m = 0.0$ per bit, and 100 individuals, we just changed $p_m = 0$ to $p_m = 0.001 \approx 0$ in order to help the algorithm to recover from the loss of an allele in a locus for the whole population.

Parameter set 1.2: Population size $N = 20$, chromosome length $l = 32$, 2-tournament selection, probability of uniform crossover $p_c = 1.0$, probability of mutation $p_m = 0.001$ per bit, stop criterion is a maximum of $100,000$ generations.

Parameter Justification: As the snake-in-the-box problem for a 5-dimensional hypercube is encoded as 32 bit string, the length of the chromosome was chosen as $l = 32$. The rest of parameters: 2-tournament selection, a probability of uniform crossover of $p_c = 1.0$, a probability of mutation $p_m = 0.001 \approx 0$, were as in parameter set 1.1

Specific Hypothesis 1.1

Equation 7.1 is satisfied for the one-max problem using the entropy metric to measure diversity, solution quality of the best solution found so far to measure expected performance, and parameter set 1.1.

Specific Hypothesis 1.2

Equation 7.1 is satisfied for the one-max problem using the entropy metric to measure diversity, number of generations to reach a global maximum to measure expected performance, and parameter set 1.1.

Specific Hypothesis 1.3

Equation 7.1 is satisfied for the one-max problem using the Hamming metric to measure diversity, solution quality of the best solution found so far to measure expected performance, and parameter set 1.1.

Specific Hypothesis 1.4

Equation 7.1 is satisfied for the one-max problem using the Hamming metric to measure diversity, number of generations to reach a global maximum to measure expected performance, and parameter set 1.1.

Specific Hypothesis 1.5

Equation 7.1 is satisfied for the snake-in-the-box problem in a 5-dimensional hypercube using the entropy metric to measure diversity, solution quality of the best solution found so far to measure expected performance, and parameter set 1.2.

Specific Hypothesis 1.6

Equation 7.1 is satisfied for the snake-in-the-box problem in a 5-dimensional hypercube using the entropy metric to measure diversity, number of generations to reach a global maximum to measure expected performance, and parameter set 1.2.

Specific Hypothesis 1.7

Equation 7.1 is satisfied for the snake-in-the-box problem in a 5-dimensional hypercube using the Hamming metric to measure diversity, solution quality of the best solution found so far to measure expected performance, and parameter set 1.2.

Specific Hypothesis 1.8

Equation 7.1 is satisfied for the snake-in-the-box problem in a 5-dimensional hypercube using the Hamming metric to measure diversity, number of generations to reach a global maximum to measure expected performance, and parameter set 1.2.

Method

Use 90 different seeds to obtain 90 initial populations with a range of diversity values maintaining the initial population size constant (20). Run a GA for each function with each initial population. Record the quality of the best solution found so far at every z generations, where $z = 5$ until generation 20 and after that at each 20 generations for the one-max problem, and $z = 10$ until generation 110 for the snake-in-the-box problem. Record the total number of generations to obtain the global maximum. In both cases the quality of the solution of the initial population is recorded.

Data Analysis

Plot data sets using plots of performance versus diversity. Check for linear correlations using Pearson's correlation coefficients. If warranted, use Fisher's R to Z transform in order to test if the correlation between two factors is near zero.

7.1.1 Results for Specific Hypothesis 1.1

The first specific hypothesis to test is the following: Equation 7.1 is satisfied for the one-max problem using the entropy metric to measure diversity, solution quality to measure expected performance, and parameter set 1.1.

Table 7.2 shows the corresponding Pearson's coeficients—for different trials until generation 60, after generation 60 a ceiling effect begins to appear—where it can be observed that there is no significant linear correlation between diversity in the initial population and the quality of the solution for these test sets (see Figures 7.1 to 7.21). As the Pearson's values are small, in order to see if the two variables, diversity and solution quality, have correlation zero, the Fisher's coefficient is presented in Table 7.3, where no values are statistically significant.

		Generation						
Trial		0	5	10	15	20	40	60
1	R_{XY}	0.099	0.134	0.198	0.174	0.088	-0.066	-0.080
2	R_{XY}	-0.084	-0.122	-0.094	-0.198	-0.047	-0.128	-0.165
3	R_{XY}	-0.048	-0.087	-0.033	0.033	0.041	0.066	0.081

Table 7.2: Pearson's Coefficient for the one-max problem. Entropy metric used. No significant linear correlation found.

		Generation						
Trial		0	5	10	15	20	40	60
1	Z	0.93	1.26	1.87	1.64	0.82	-0.62	-0.75
2	Z	-0.79	-1.14	-0.88	-1.87	-0.44	-1.20	-1.55
3	Z	-0.45	-0.81	-0.31	0.31	0.38	0.62	0.76

Table 7.3: Fisher's Coefficient for the one-max problem. No significant linear correlation found.

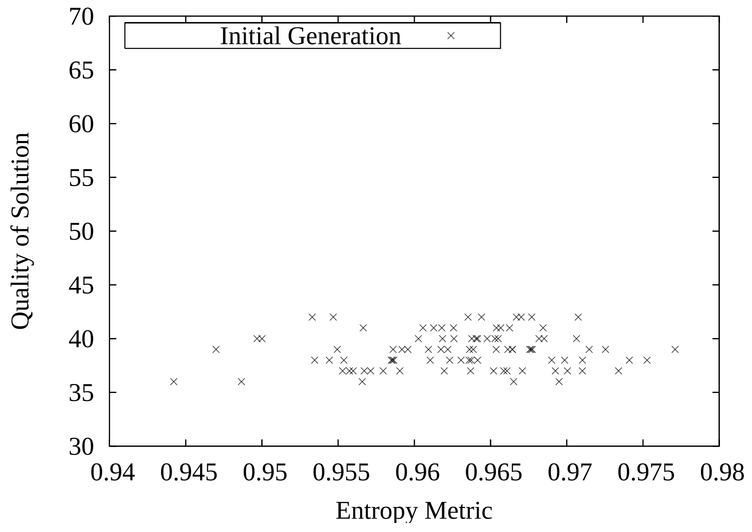


Figure 7.1: Entropy vs. quality of the solution. One-max problem. Snapshot at initial generation. Trial 1. **90** Runs.

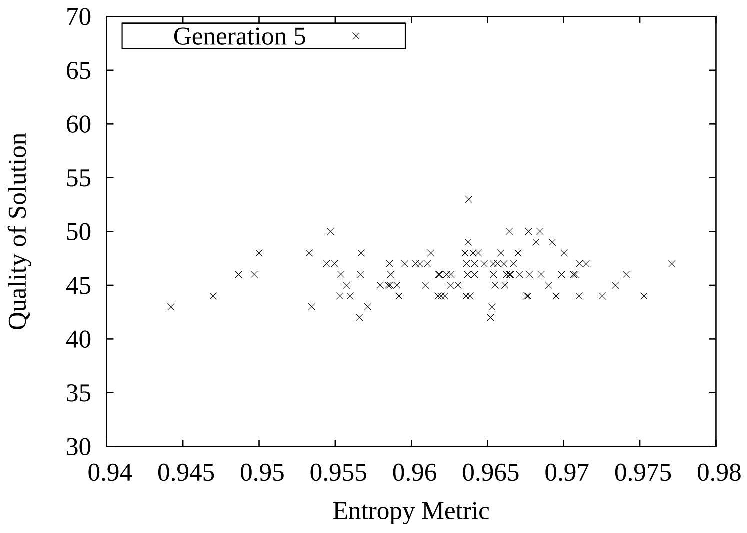


Figure 7.2: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 5. Trial 1. **90** Runs.

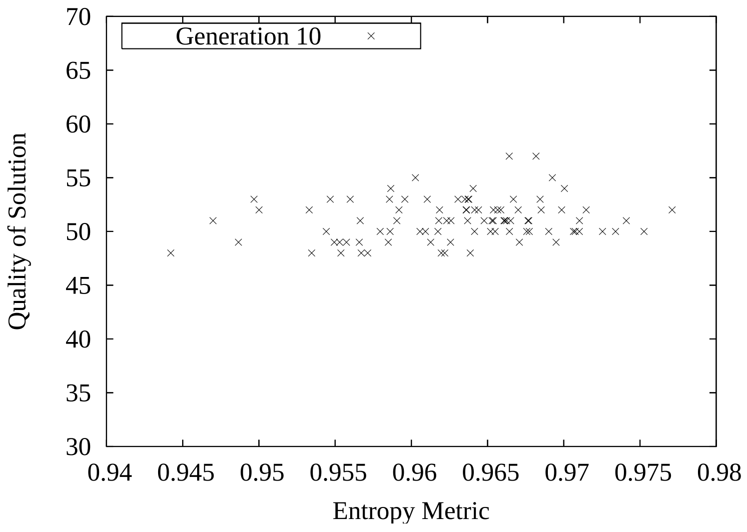


Figure 7.3: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 10. Trial 1. **90** Runs.

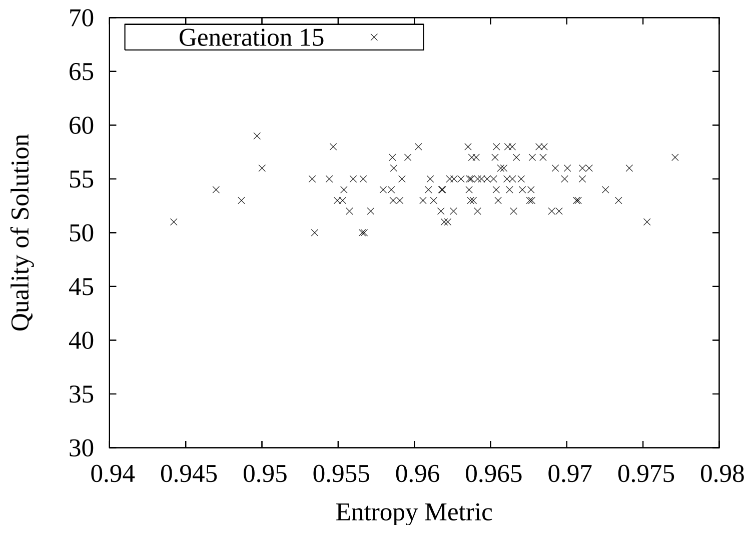


Figure 7.4: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 15. Trial 1. $\mathbf{90}$ Runs.

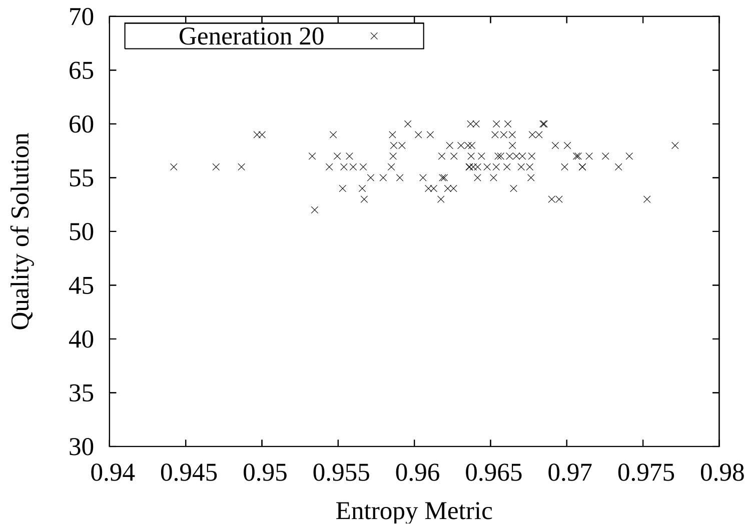


Figure 7.5: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 20. Trial 1. $\mathbf{90}$ Runs.

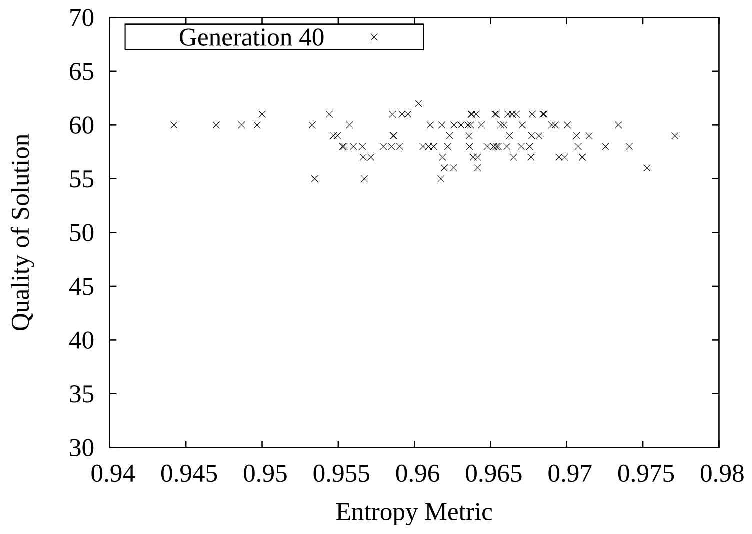


Figure 7.6: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 40. Trial 1. **90** Runs.

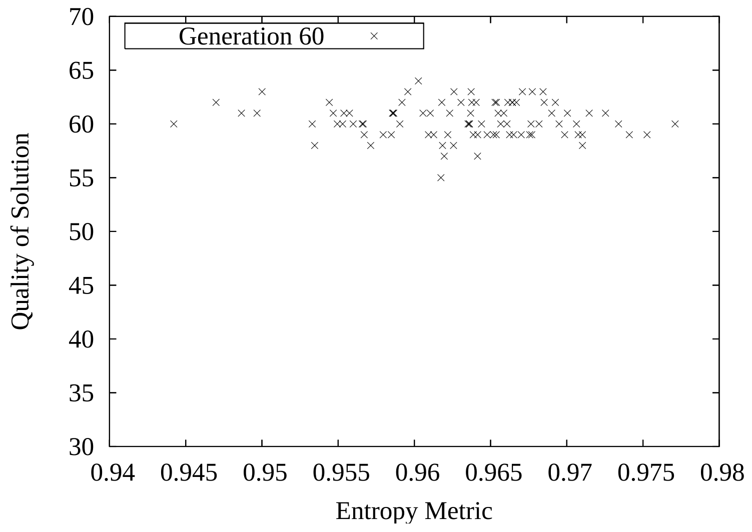


Figure 7.7: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 60. Trial 1. **90** Runs.

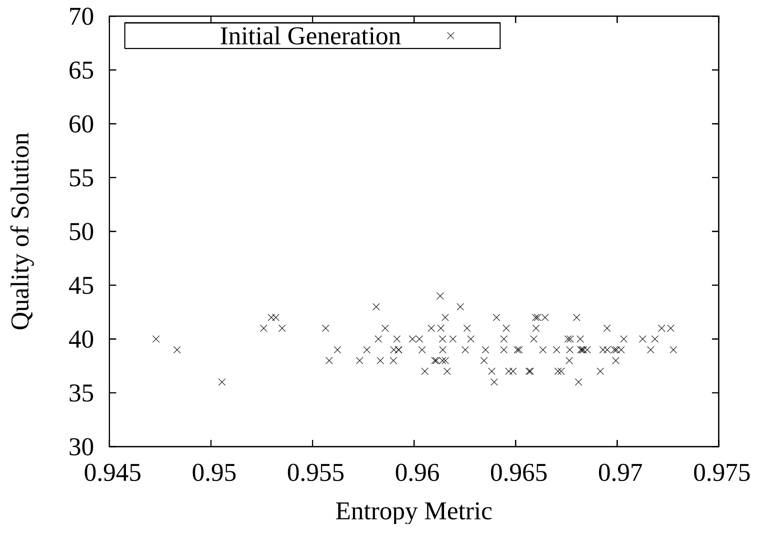


Figure 7.8: Entropy vs. quality of the solution. One-max problem. Snapshot at initial generation. Trial 2. **90** Runs.

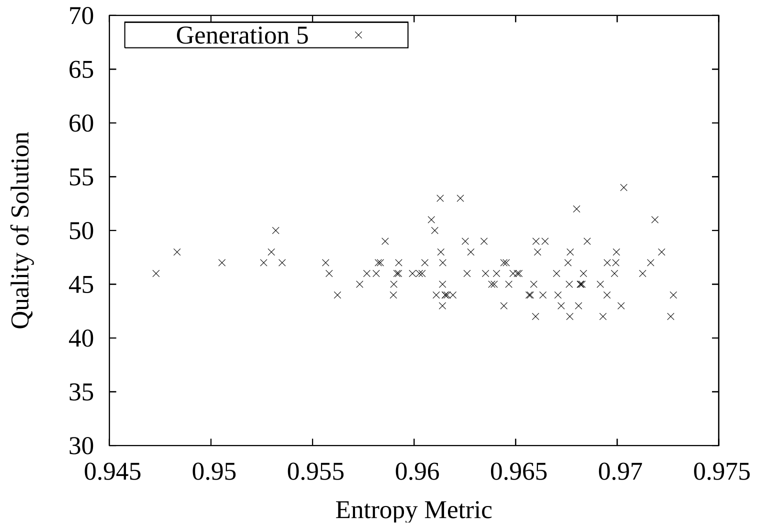


Figure 7.9: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 5. Trial 2. **90** Runs.

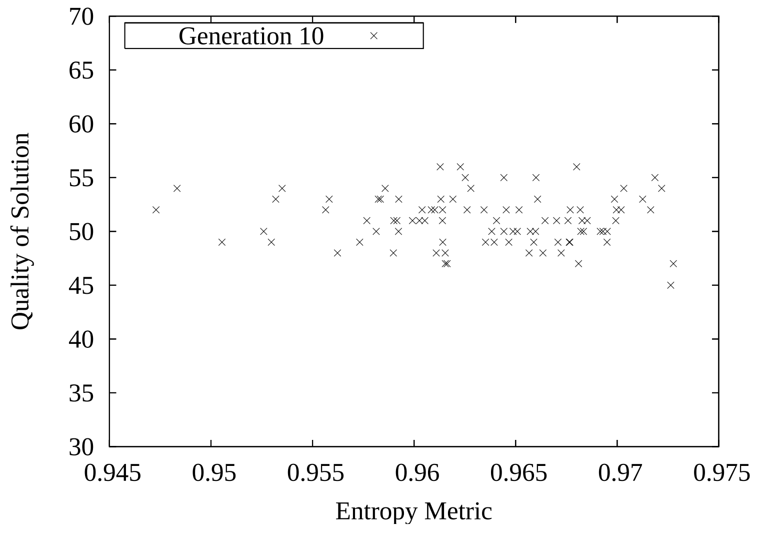


Figure 7.10: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 10. Trial 2. **90** Runs.

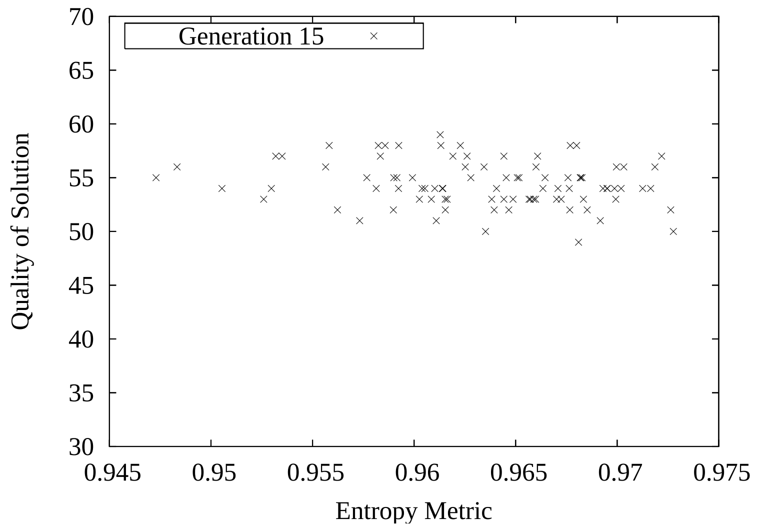


Figure 7.11: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 15. Trial 2. **90** Runs.

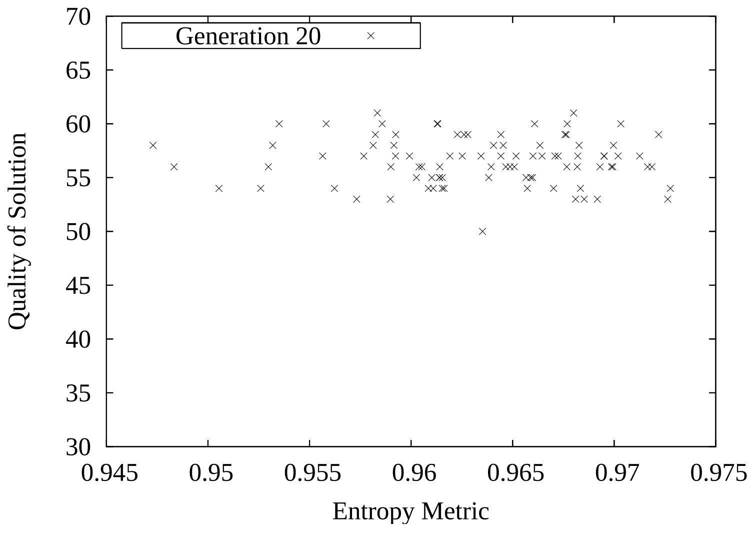


Figure 7.12: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 20. Trial 2. **90** Runs.

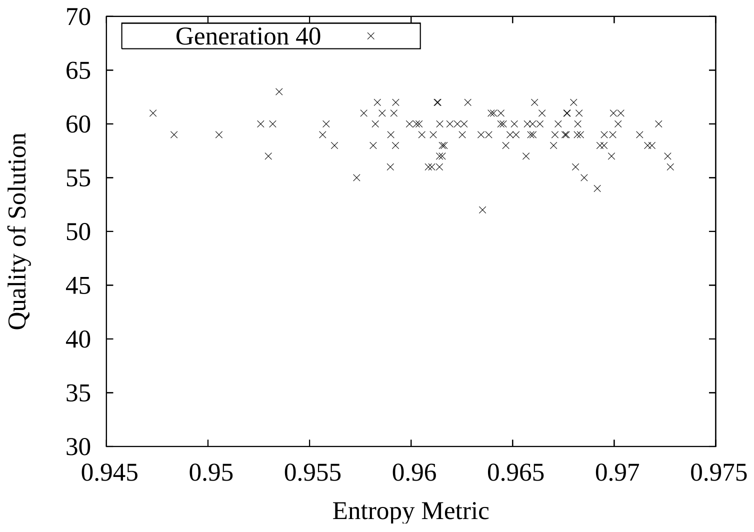


Figure 7.13: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 40. Trial 2. **90** Runs.

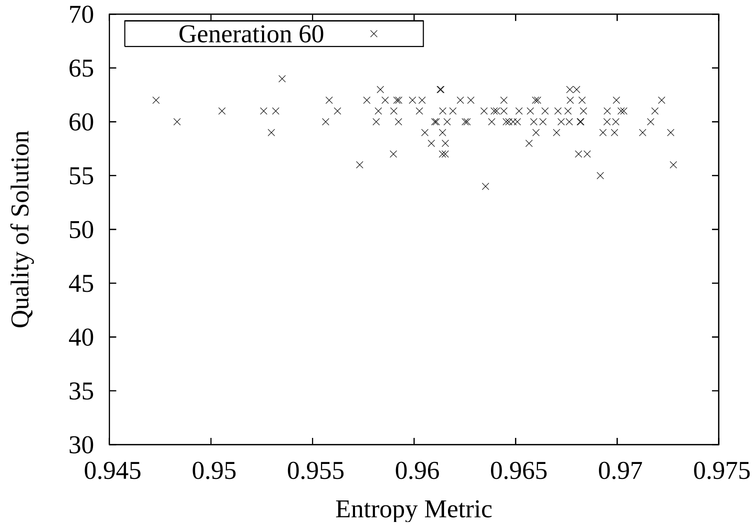


Figure 7.14: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 60. Trial 2. **90** Runs.

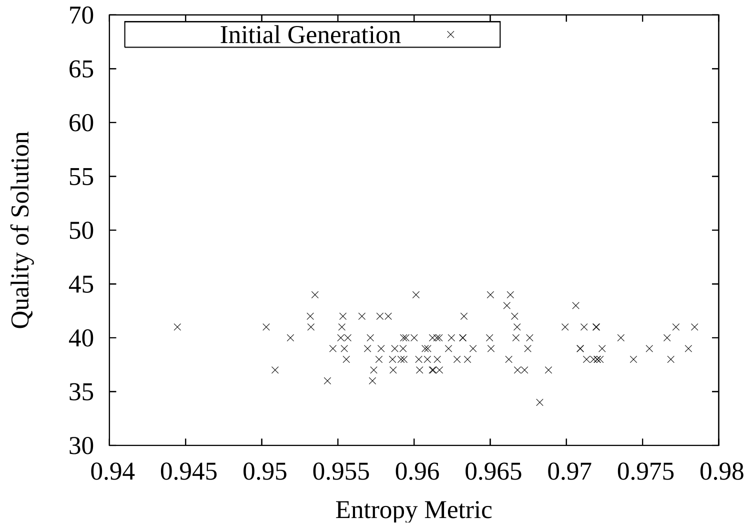


Figure 7.15: Entropy vs. quality of the solution. One-max problem. Snapshot at initial generation. Trial 3. **90** Runs.

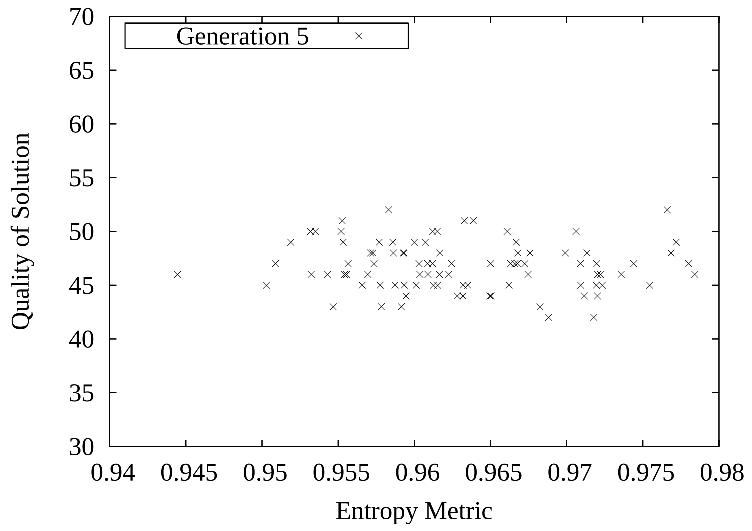


Figure 7.16: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 5. Trial 3. $\mathbf{90}$ Runs.

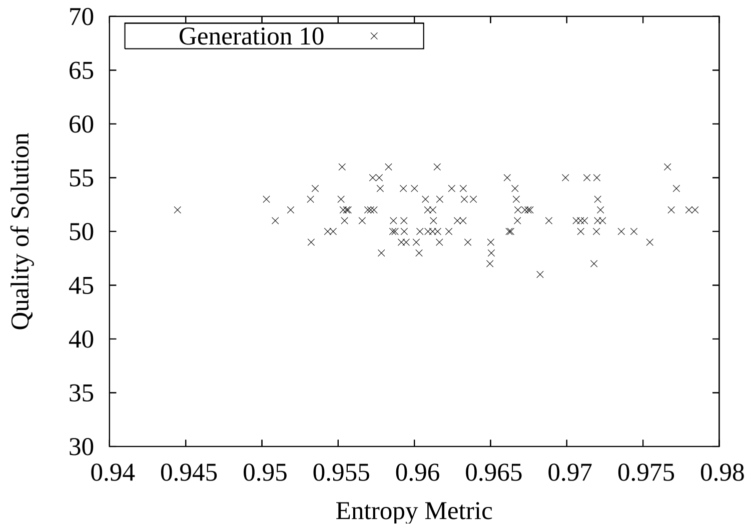


Figure 7.17: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 10. Trial 3. $\mathbf{90}$ Runs.

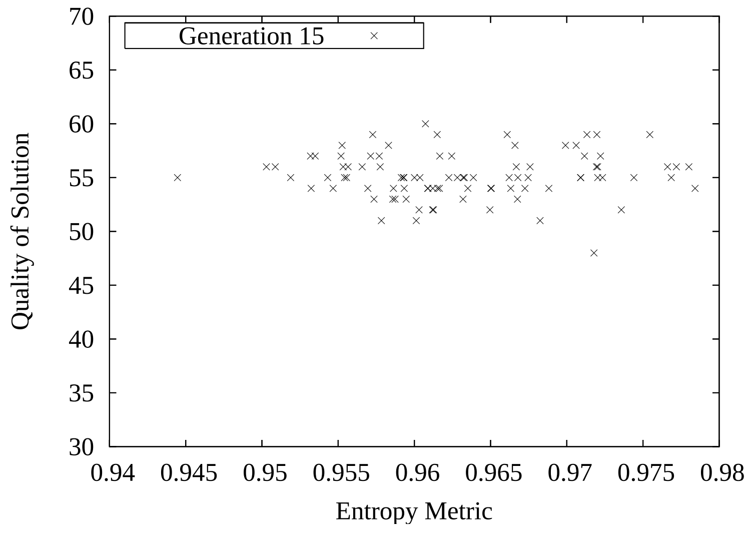


Figure 7.18: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 15. Trial 3. **90** Runs.

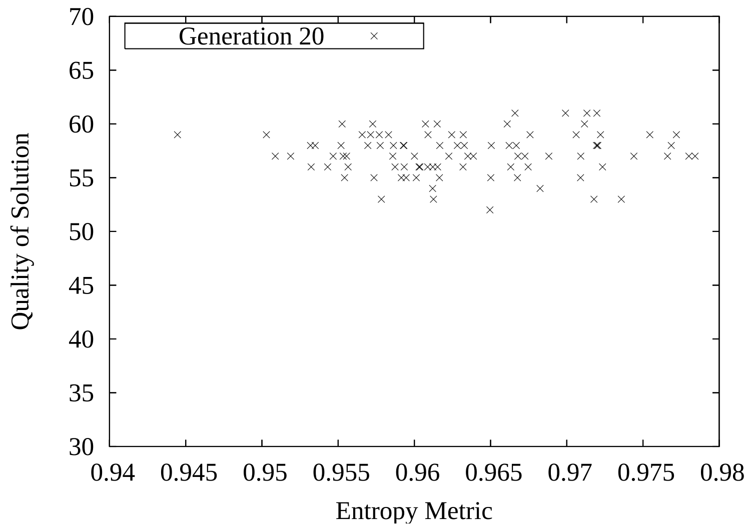


Figure 7.19: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 20. Trial 3. **90** Runs.

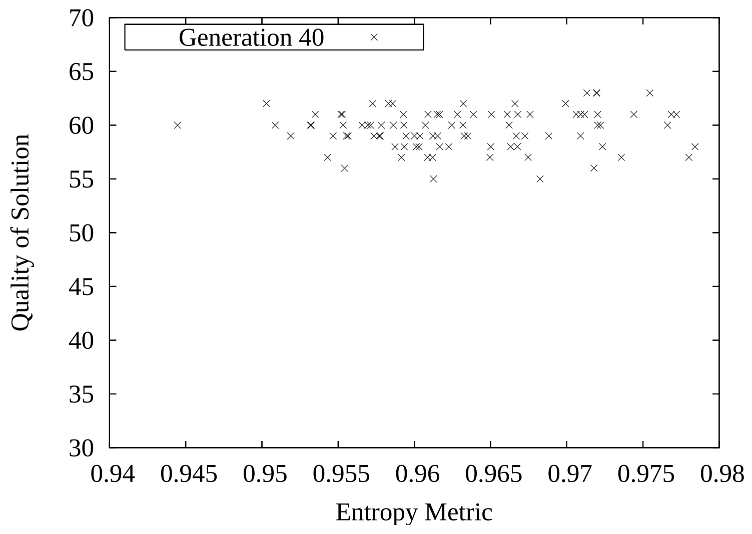


Figure 7.20: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 40. Trial 3. **90** Runs.

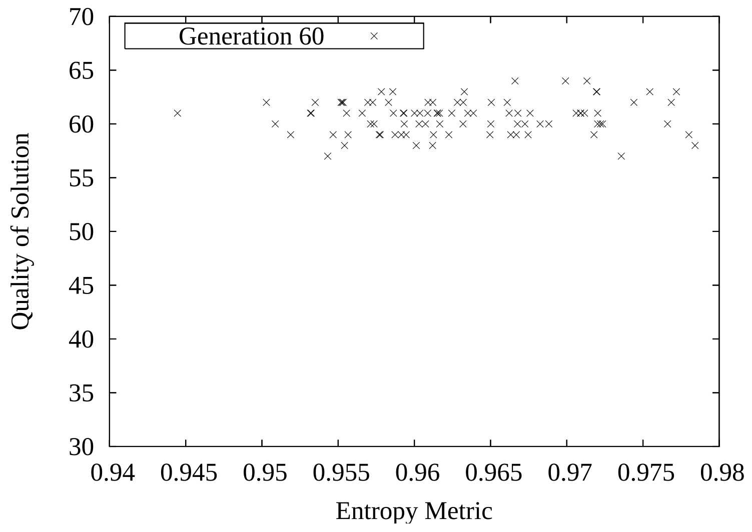


Figure 7.21: Entropy vs. quality of the solution. One-max problem. Snapshot at generation 60. Trial 3. **90** Runs.

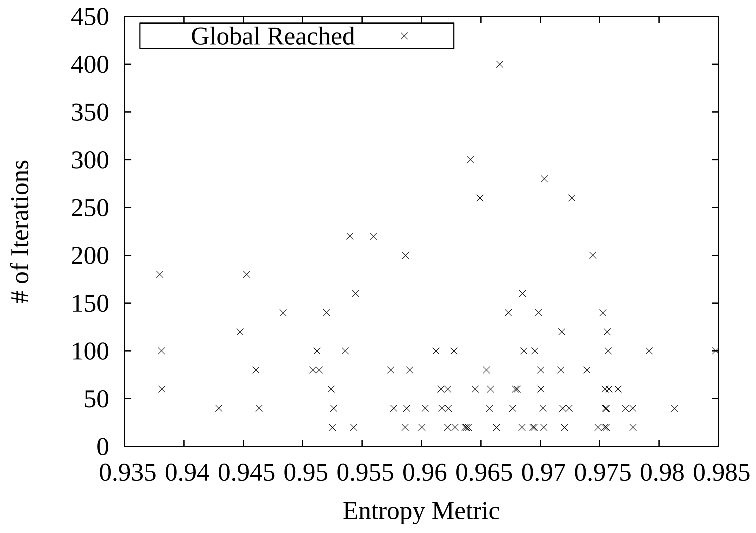


Figure 7.22: Entropy metric vs. number of generations to reach the global maximum. One-max problem. 90 Runs.

7.1.2 Results for Specific Hypothesis 1.2

The second specific hypothesis to test is the following: Equation 7.1 is satisfied for the one-max problem using the entropy metric to measure diversity, number of generations to reach a global maximum to measure expected performance, and parameter set 1.1.

Figure 7.22 takes into account the number of generations to reach a global maximum. No statistically significant correlation was discovered between the entropy metric and the expected number of generations. The Pearson's and Fisher's coefficient were of -0.106 and -0.992 (see Figure 7.22).

7.1.3 Results for Specific Hypothesis 1.3

The third specific hypothesis to test is the following: Equation 7.1 is satisfied for the one-max problem using the Hamming metric to measure diversity, solution quality to measure expected performance, and parameter set 1.1.

Table 7.4 shows the corresponding Pearson's coeficients (for the different trials) were it can be observed that there is no significant linear correlation between diversity in the initial population and the quality of the solution for these test sets (see Figures 7.23 to 7.43). As the Pearson's values are small, in order to see if the two variables, diversity and

		Generation						
Trial		0	5	10	15	20	40	60
1	R_{XY}	0.092	0.135	0.203	0.177	0.088	−0.064	−0.077
2	R_{XY}	−0.078	−0.118	−0.089	−0.196	−0.046	−0.121	−0.159
3	R_{XY}	−0.050	−0.092	−0.035	0.033	0.040	0.065	0.082

Table 7.4: Pearson's Coefficient for the one-max problem. Hamming metric used. No linear correlation found.

		Generation						
Trial		0	5	10	15	20	40	60
1	Z	0.86	1.27	1.92	1.67	0.82	−0.60	−0.72
2	Z	−0.73	−1.11	−0.83	−1.85	−0.43	−1.13	−1.50
3	Z	−0.47	−0.86	−0.33	0.31	0.37	0.61	0.77

Table 7.5: Fisher's Coefficient for the one-max problem. Hamming metric used. Zero correlation found.

solution quality, have correlation zero, the Fisher's coefficient is presented in Table 7.5, where no values are statistically significant.

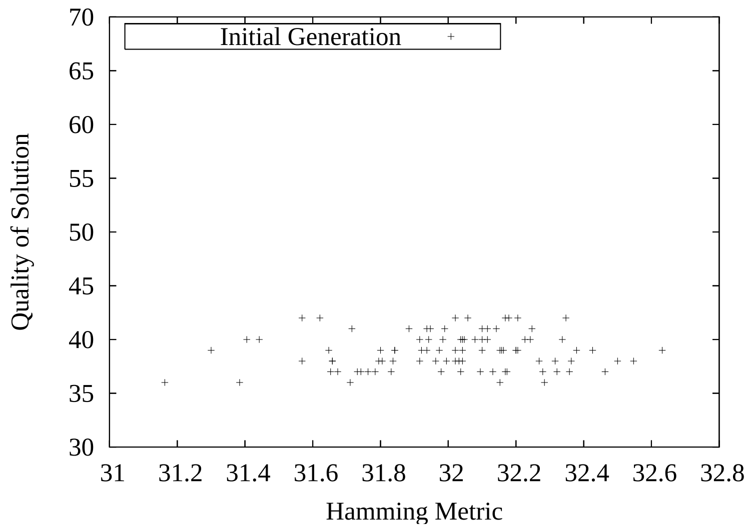


Figure 7.23: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at initial generation. Trial 1. 90 Runs.

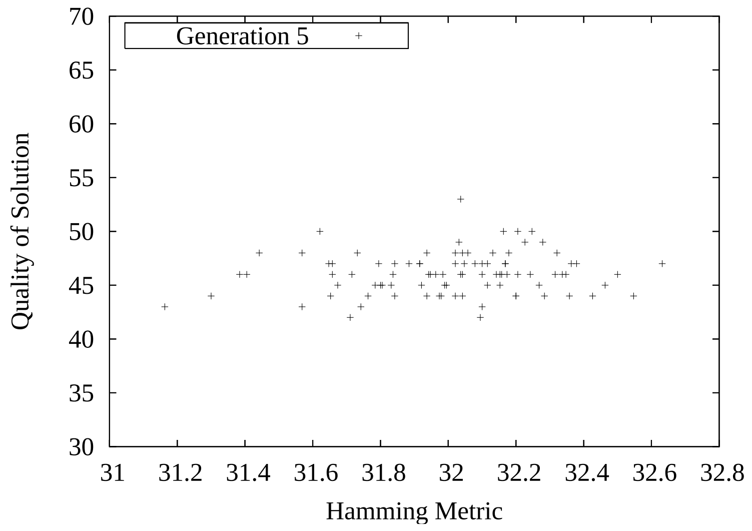


Figure 7.24: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 5. Trial 1. **90** Runs.

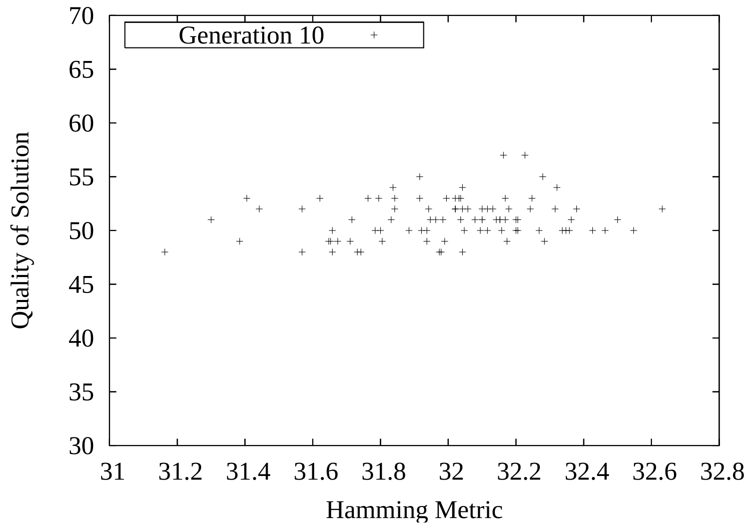


Figure 7.25: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 10. Trial 1. **90** Runs.

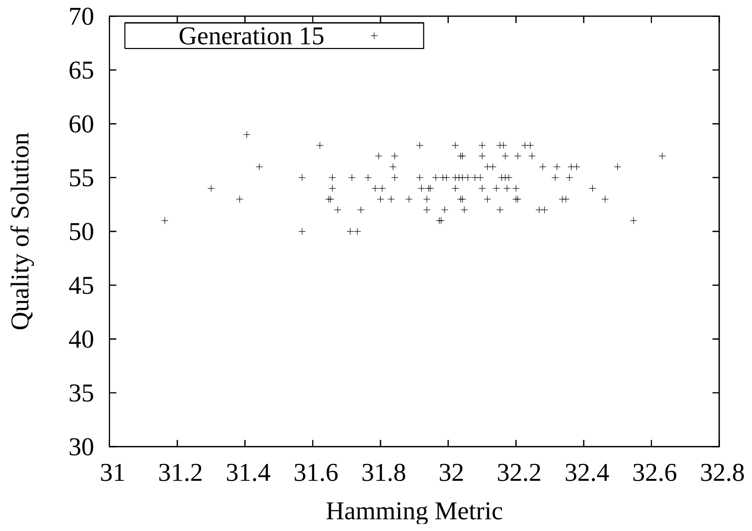


Figure 7.26: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 15. Trial 1. 90 Runs.

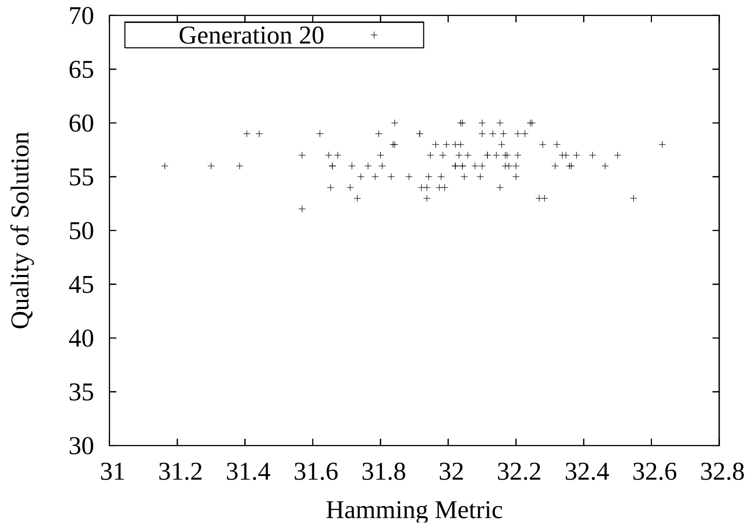


Figure 7.27: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 20. Trial 1. 90 Runs.

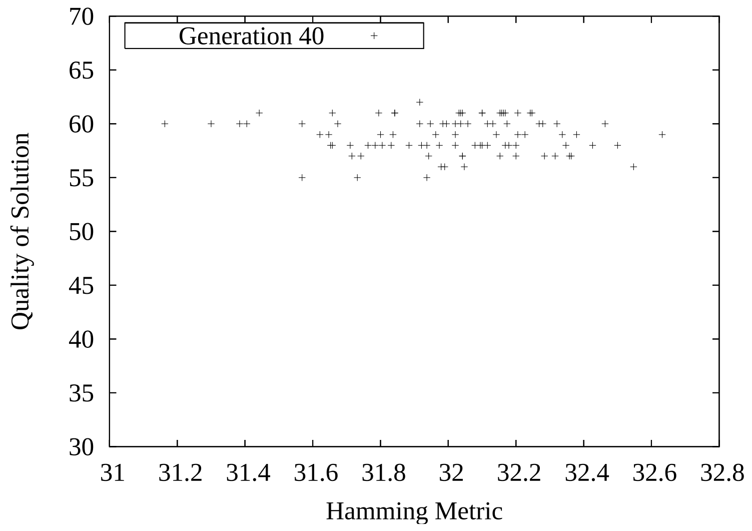


Figure 7.28: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 40. Trial 1. 90 Runs.

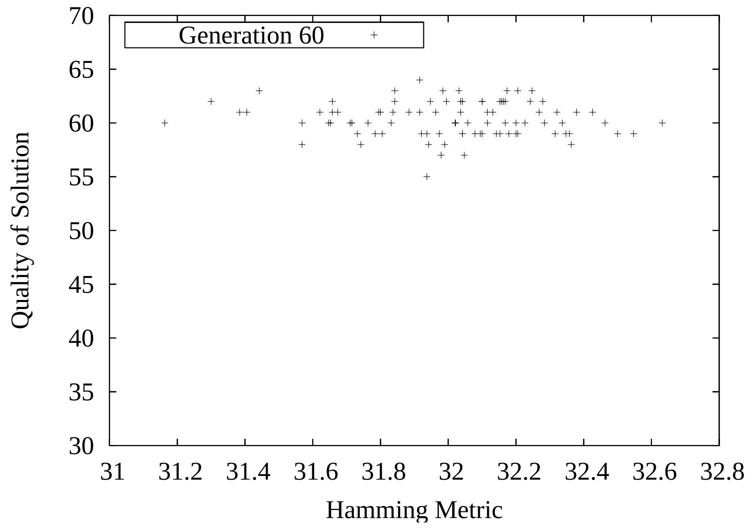


Figure 7.29: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 60. Trial 1. 90 Runs.

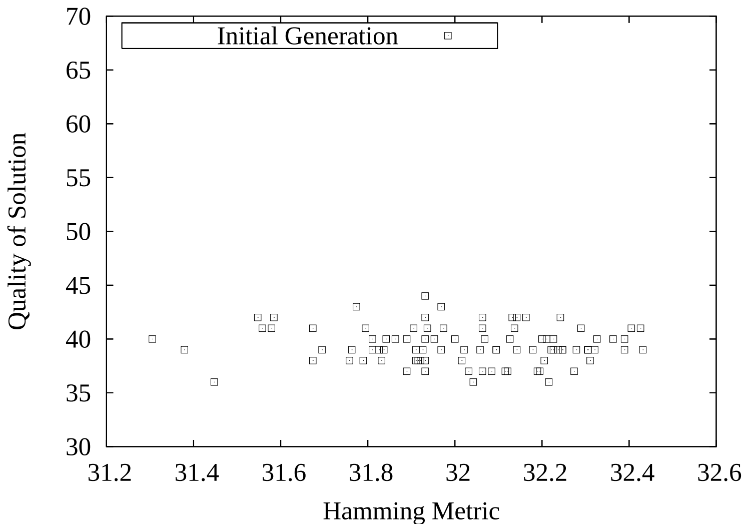


Figure 7.30: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at initial generation. Trial 2. 90 Runs.

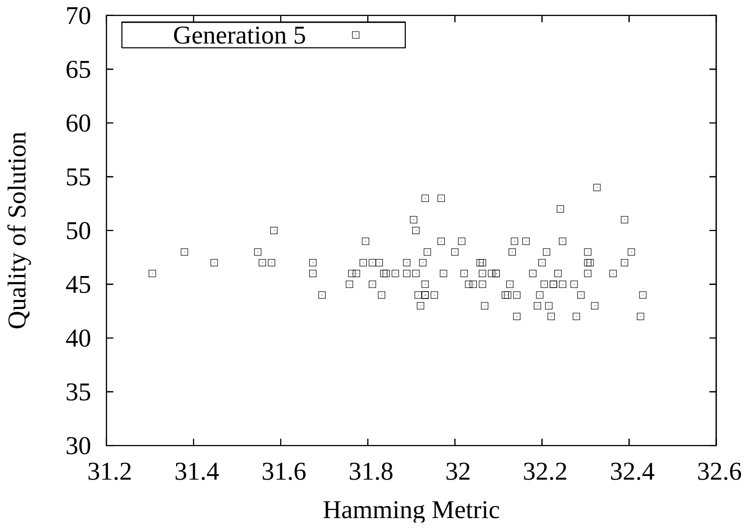


Figure 7.31: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 5. Trial 2. 90 Runs.

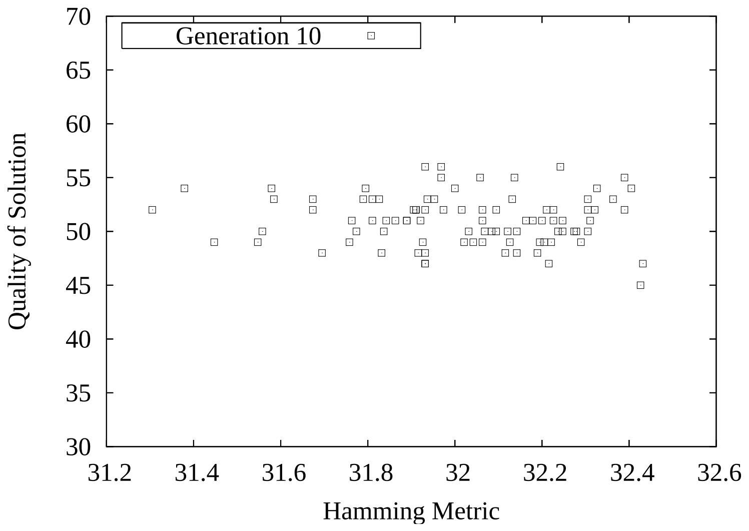


Figure 7.32: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 10. Trial 2. **90** Runs.

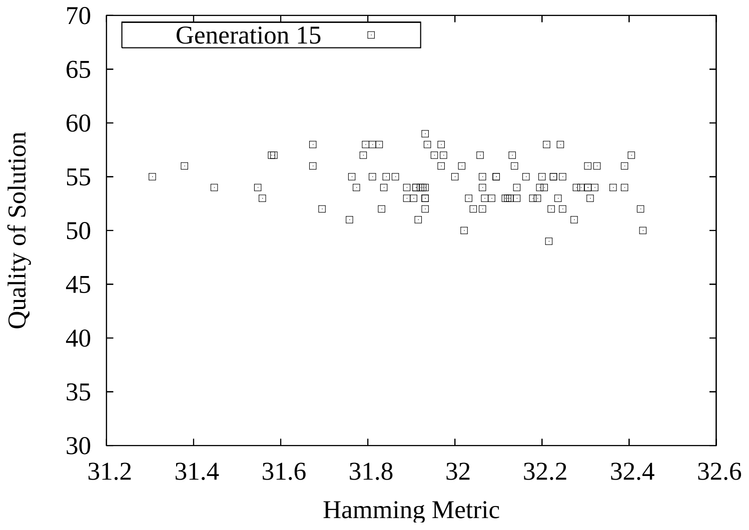


Figure 7.33: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 15. Trial 2. **90** Runs.

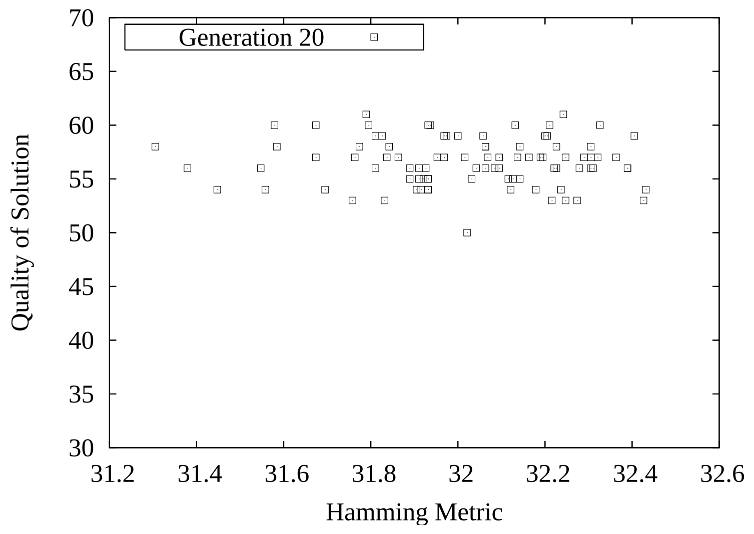


Figure 7.34: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 20. Trial 2. $\mathbf{90}$ Runs.

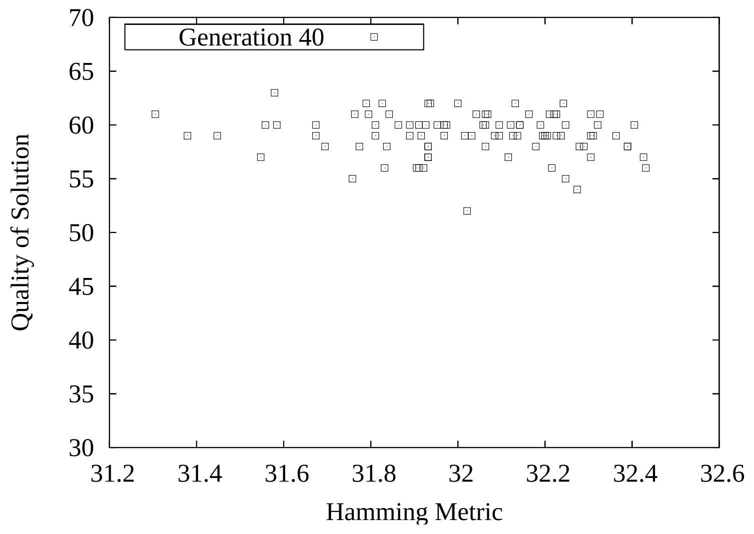


Figure 7.35: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 40. Trial 2. $\mathbf{90}$ Runs.

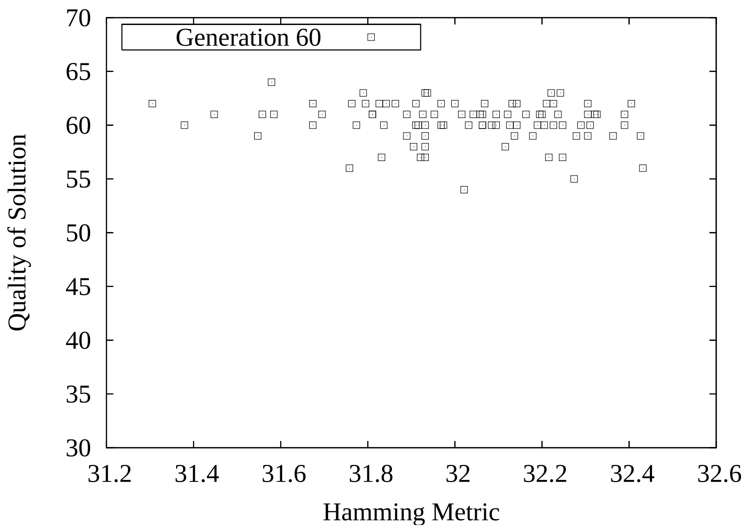


Figure 7.36: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 60. Trial 2. $\mathbf{90}$ Runs.

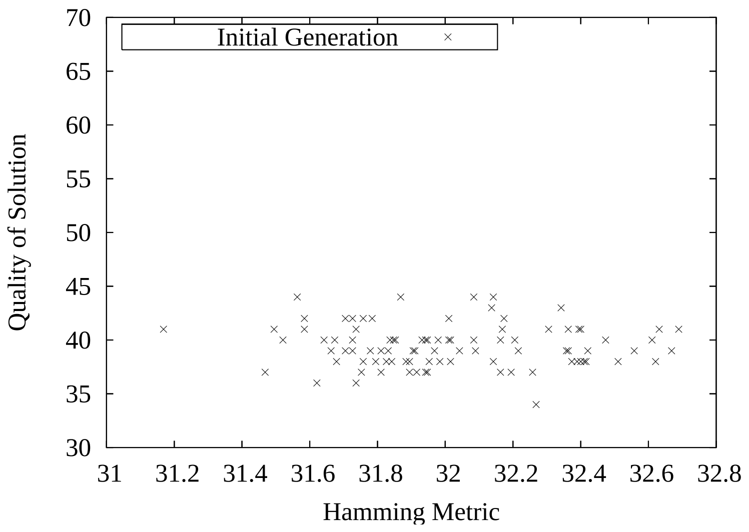


Figure 7.37: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at initial generation. Trial 3. $\mathbf{90}$ Runs.

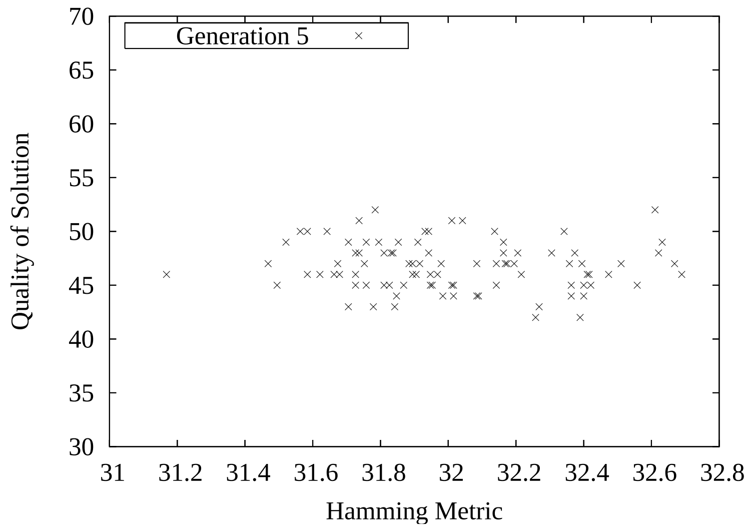


Figure 7.38: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 5. Trial 3. $\mathbf{90}$ Runs.

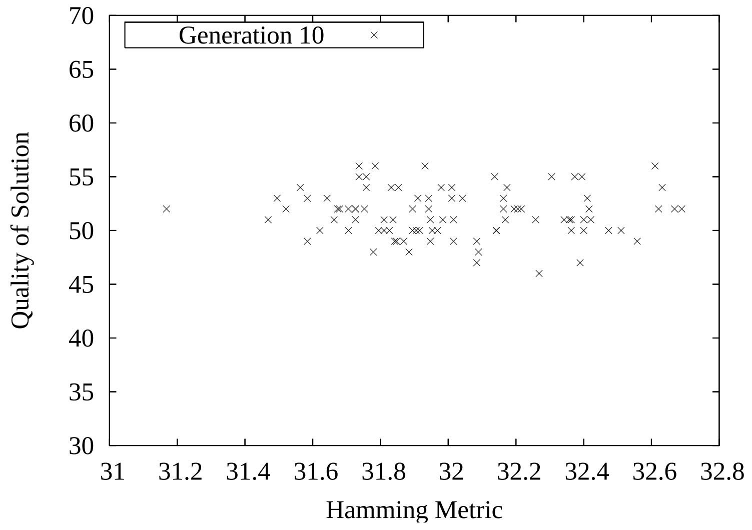


Figure 7.39: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 10. Trial 3. $\mathbf{90}$ Runs.

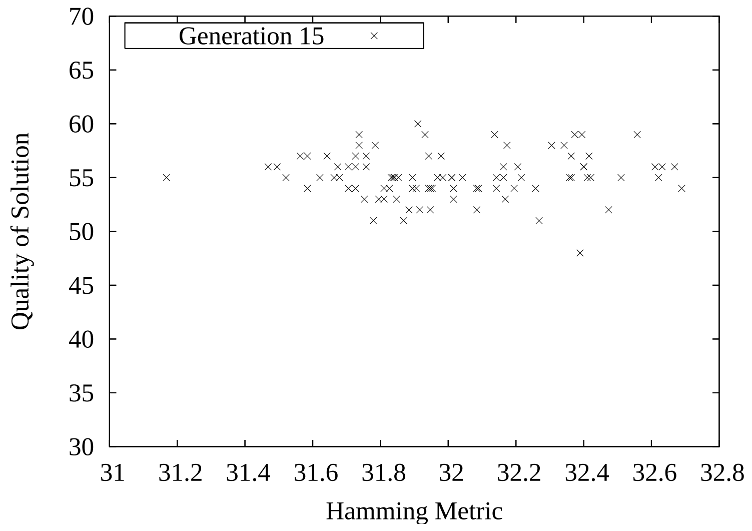


Figure 7.40: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 15. Trial 3. **90** Runs.

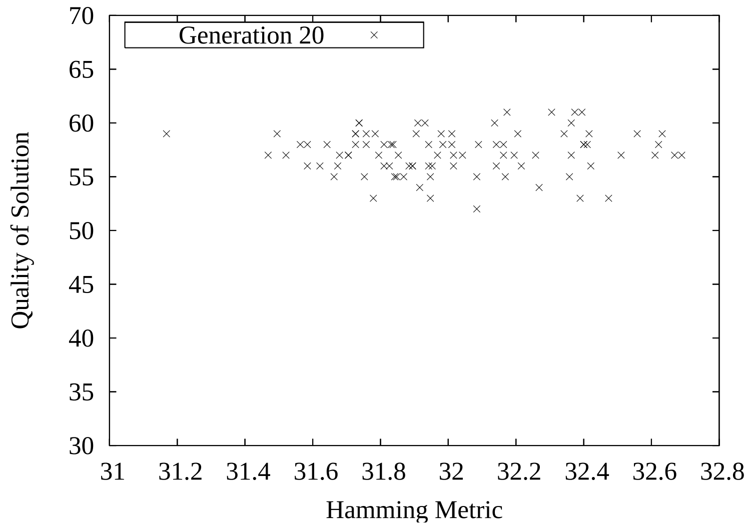


Figure 7.41: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 20. Trial 3. **90** Runs.

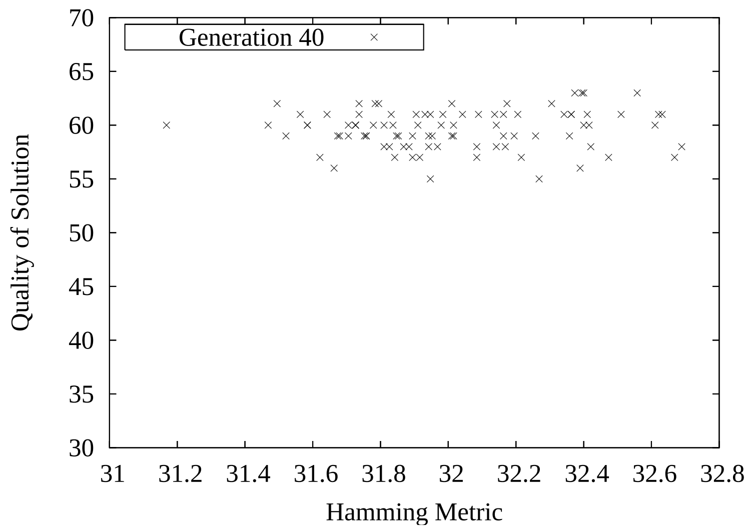


Figure 7.42: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 40. Trial 3. **90** Runs.

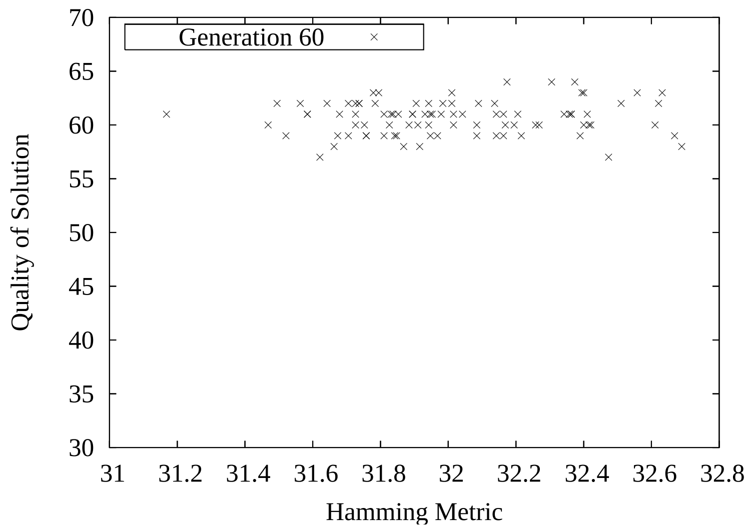


Figure 7.43: Hamming metric vs. quality of the solution. One-max Problem. Snapshot at generation 100. Trial 3. **90** Runs.

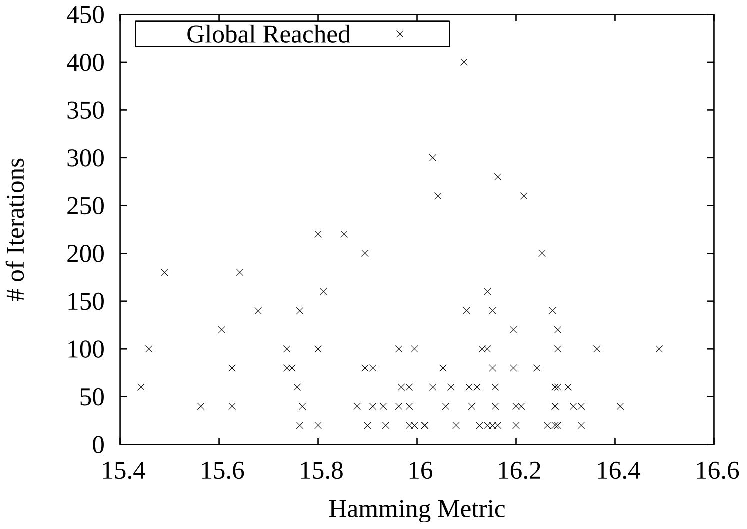


Figure 7.44: Hamming metric vs. number of generations to reach the global maximum. One-max Problem. $\mathbf{90}$ Runs.

7.1.4 Results for Specific Hypothesis 1.4

The fourth specific hypothesis to test is the following: Equation 7.1 is satisfied for the one-max problem using the Hamming metric to measure diversity, number of generations to reach a global maximum to measure expected performance, and parameter set 1.1.

Figure 7.44 takes into account the number of generations to reach a global maximum. There is no statistically significant correlation was discovered between the Hamming metric and the expected number of generations as the Pearson's and Fisher's coefficient have values of -0.096 and -0.898, which shows that diversity and solution quality are independent.

7.1.5 Results for Specific Hypothesis 1.5

The fifth specific hypothesis to test is the following: Equation 7.1 is satisfied for the snake-in-the-box problem using the entropy metric to measure diversity, solution quality to measure expected performance, and parameter set 1.2

No statistically significant correlation between the entropy metric and the expected quality of the solution was discovered as the Pearson and Fisher's coefficients show in Tables 7.6 and 7.7 that correspond to the snapshots as in Figures 7.45 to 7.80.

		Generation					
Trial		0	10	20	30	40	50
1	R_{XY}	-0.068	-0.014	-0.008	-0.017	-0.022	-0.015
2	R_{XY}	0.185	-0.115	-0.115	-0.186	-0.143	-0.150
3	R_{XY}	-0.015	-0.047	-0.003	0.045	0.028	0.028
Trial		60	70	80	90	100	110
1	R_{XY}	-0.039	-0.039	-0.011	0.005	0.016	0.006
2	R_{XY}	-0.154	-0.108	-0.097	-0.156	-0.161	-0.227
3	R_{XY}	0.073	0.103	0.103	0.122	0.119	0.127

Table 7.6: Pearson's Coefficients for the Snake-in-the-box problem. Entropy metric. No linear correlation found.

		Generation											
Trial		0	10	20	30	40	50	60	70	80	90	100	110
1	Z	-0.64	-0.13	-0.07	-0.16	-0.21	-0.14	-0.36	-0.36	-0.10	0.05	0.15	0.06
2	Z	1.75	-1.08	-1.08	-1.76	-1.34	-1.41	-1.45	-1.01	-0.91	-1.47	-1.51	-2.15
3	Z	-0.14	-0.44	-0.03	0.42	0.26	0.26	0.68	0.96	0.96	1.44	1.11	1.19

Table 7.7: Fisher's Coefficients for the Snake-in-the-box problem. Entropy metric. No linear correlation found.

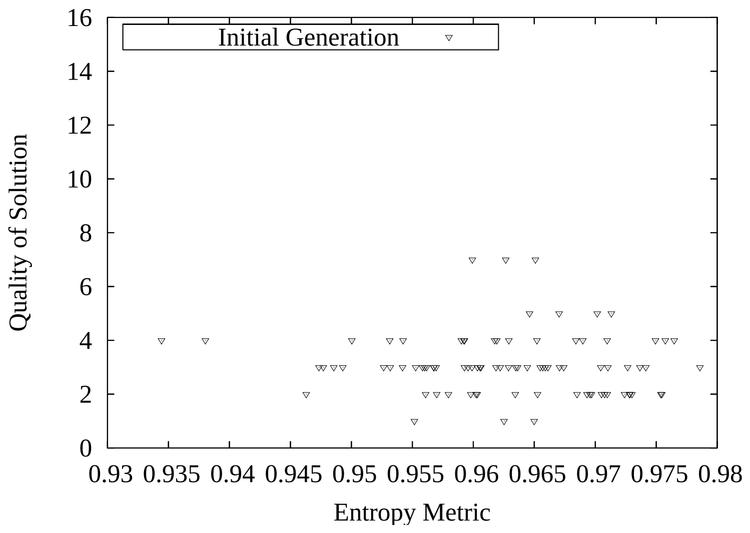


Figure 7.45: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at initial generation. 90 Runs.

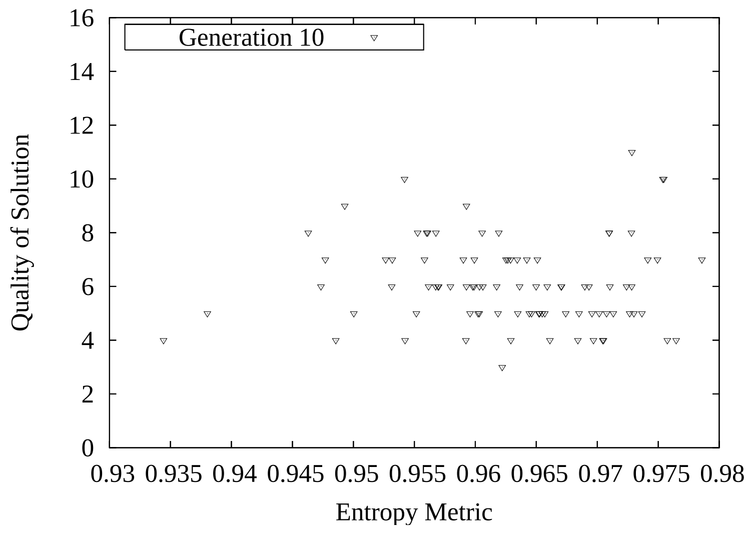


Figure 7.46: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 10. **90** Runs.

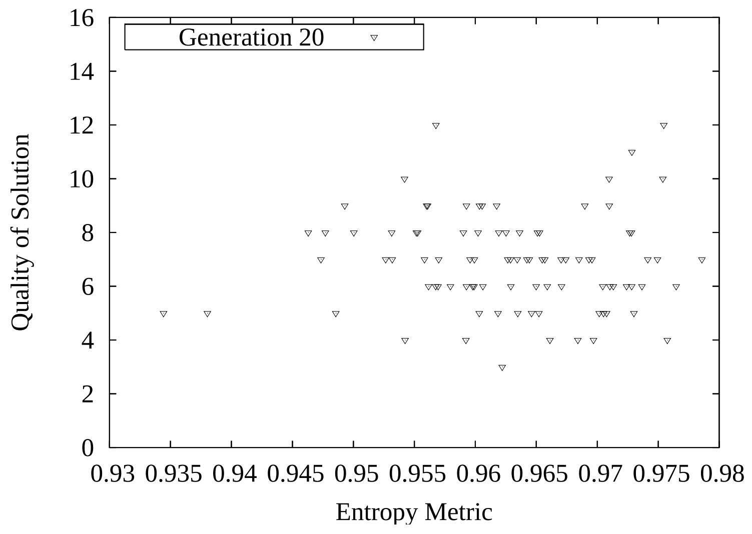


Figure 7.47: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 20. **90** Runs.

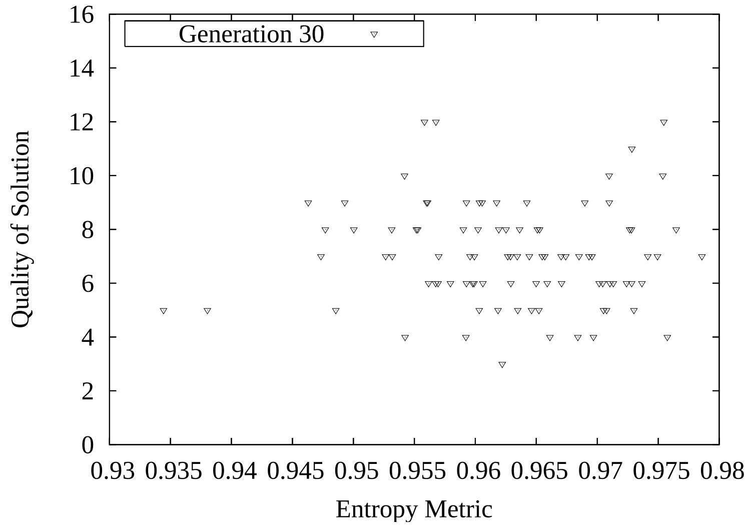


Figure 7.48: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 30. **90** Runs.

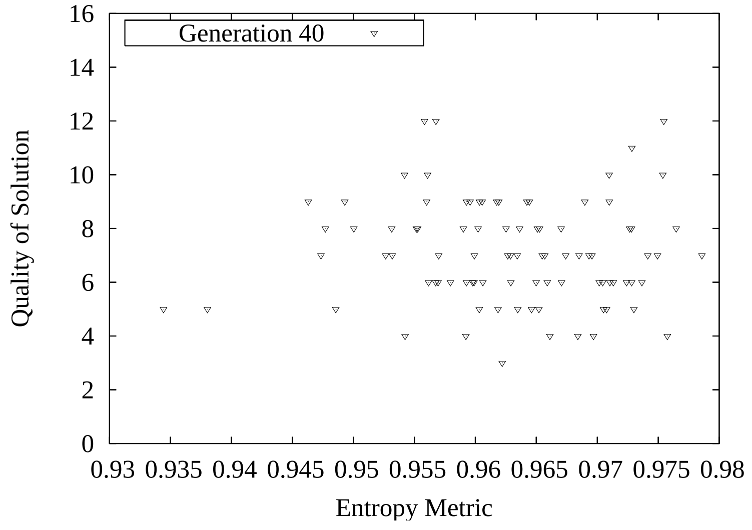


Figure 7.49: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 40. **90** Runs.

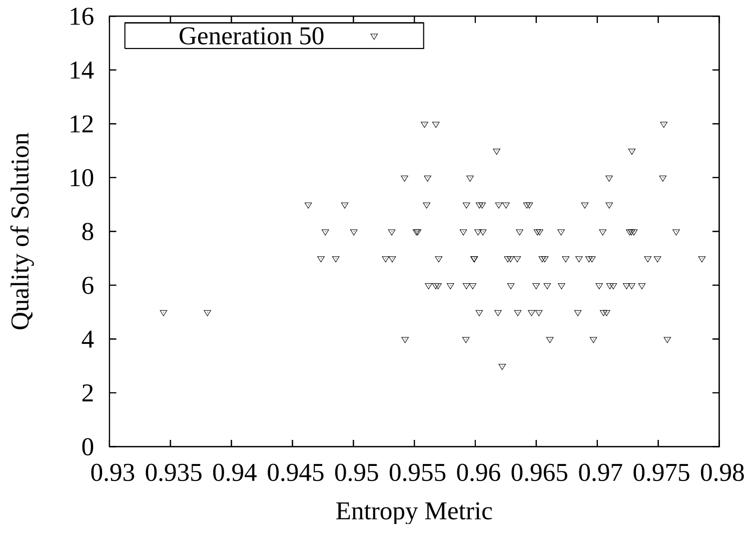


Figure 7.50: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 50. **90** Runs.

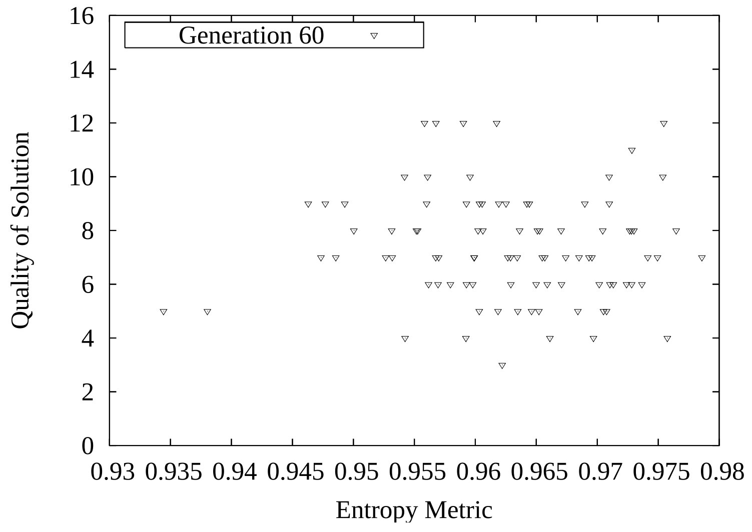


Figure 7.51: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 60. **90** Runs.

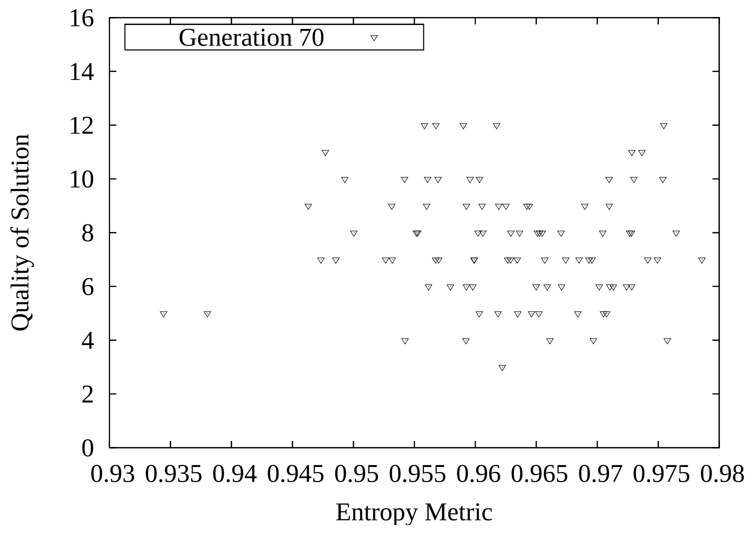


Figure 7.52: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 70. **90** Runs.

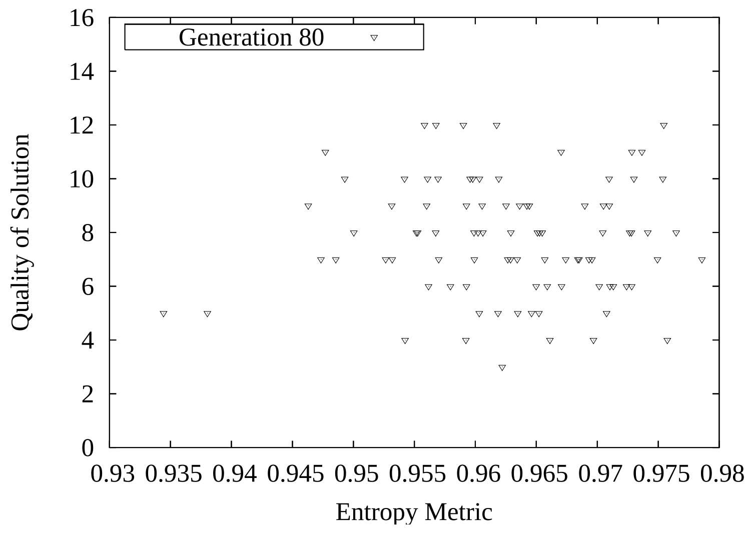


Figure 7.53: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 80. **90** Runs.

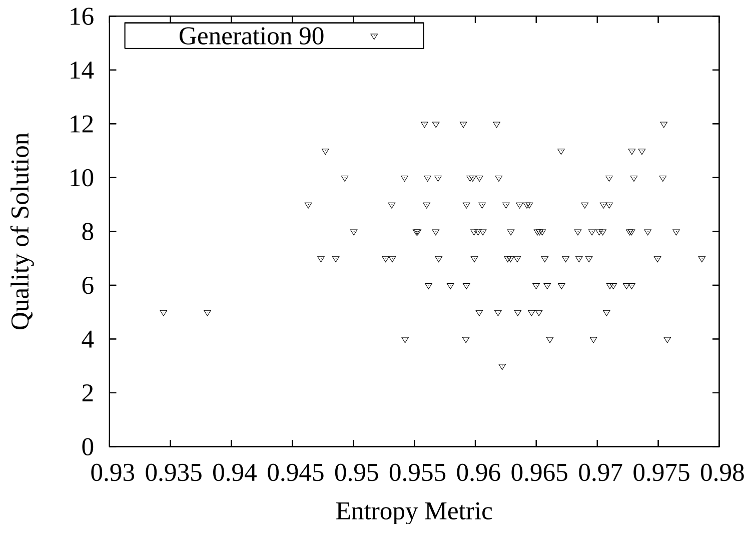


Figure 7.54: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 90. **90** Runs.

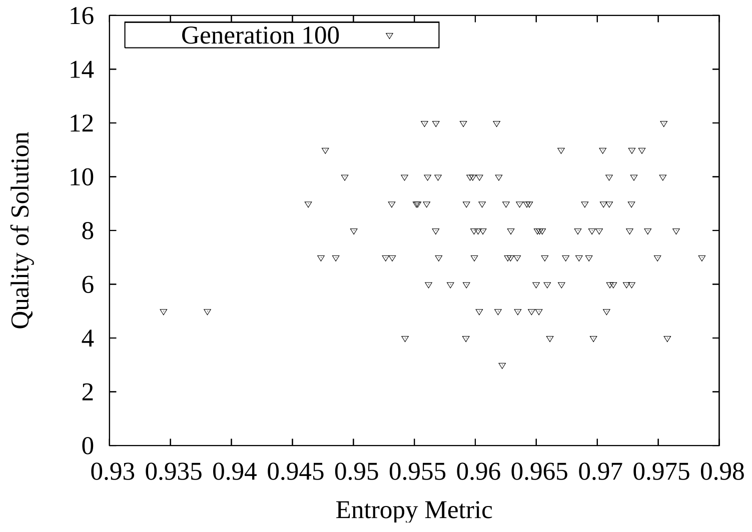


Figure 7.55: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 100. **90** Runs.

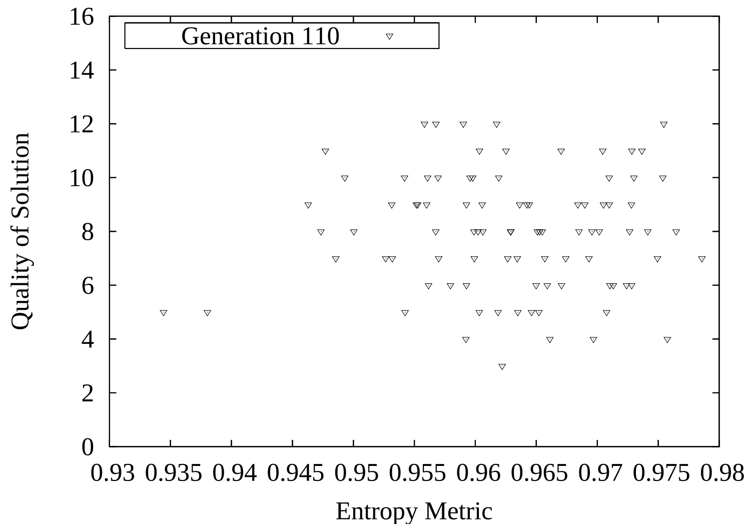


Figure 7.56: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 110. **90** Runs.

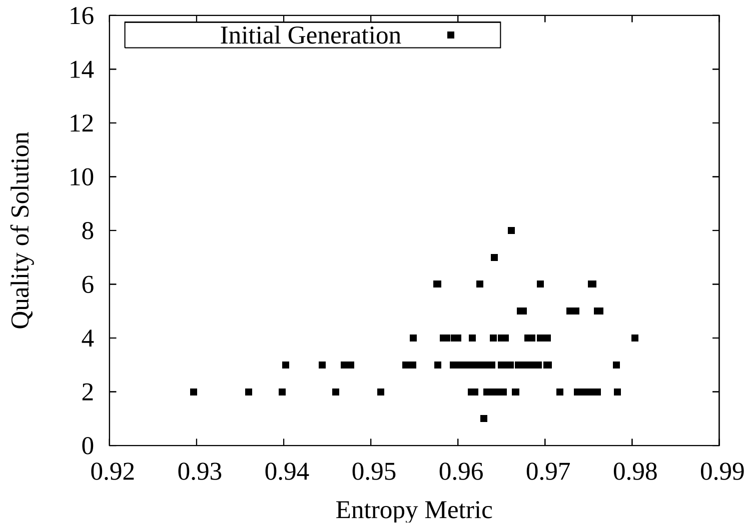


Figure 7.57: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at initial generation. **90** Runs.

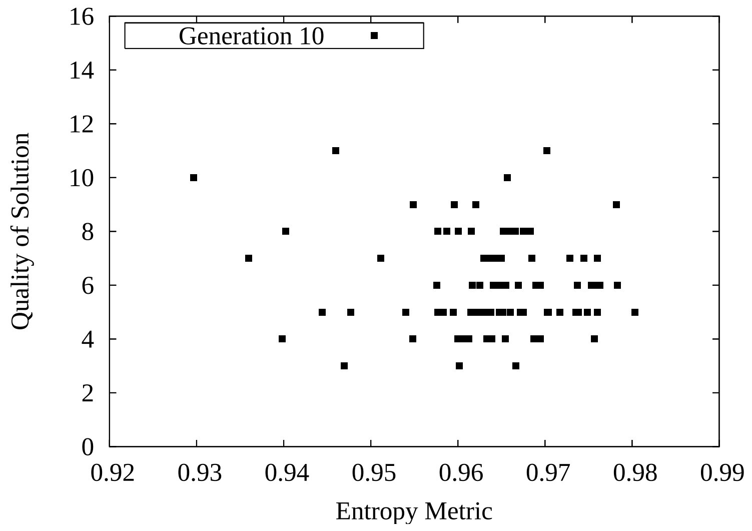


Figure 7.58: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 10. **90** Runs.

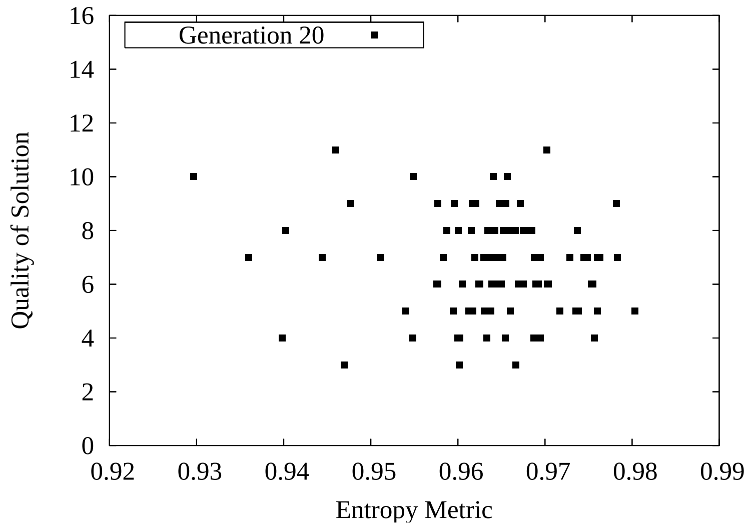


Figure 7.59: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 20. **90** Runs.

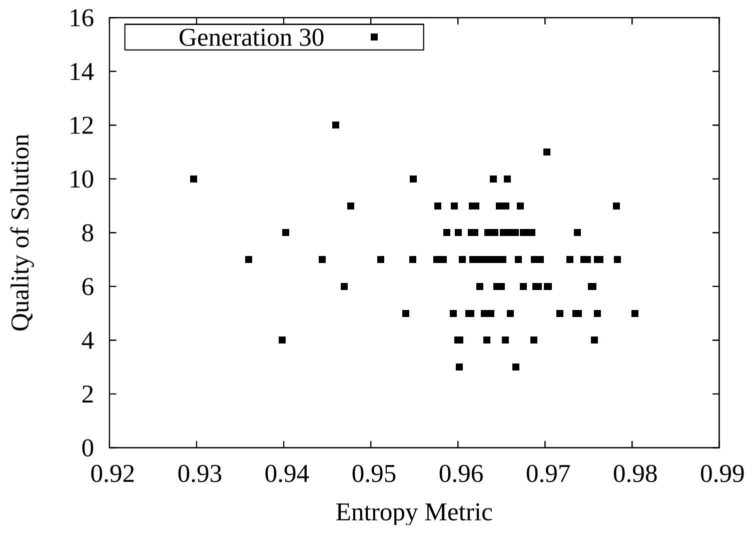


Figure 7.60: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 30. **90** Runs.

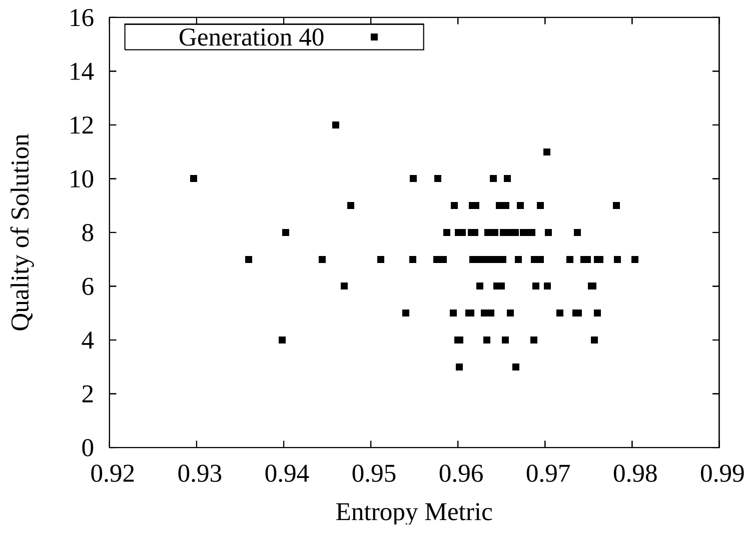


Figure 7.61: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 40. **90** Runs.

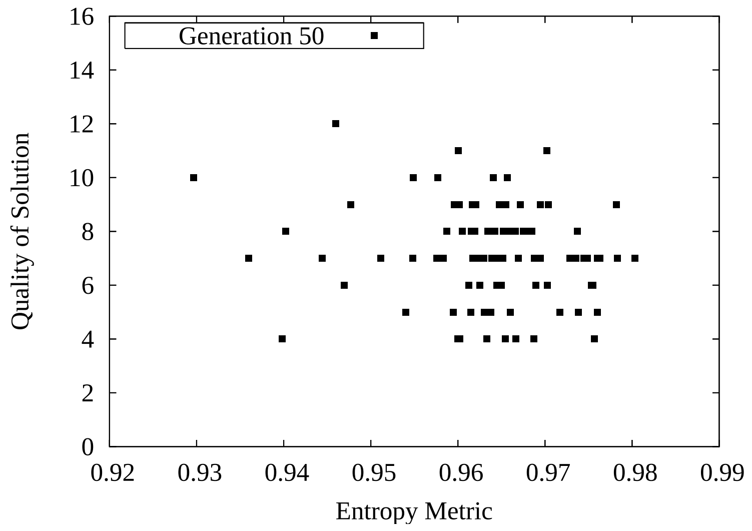


Figure 7.62: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 50. **90** Runs.

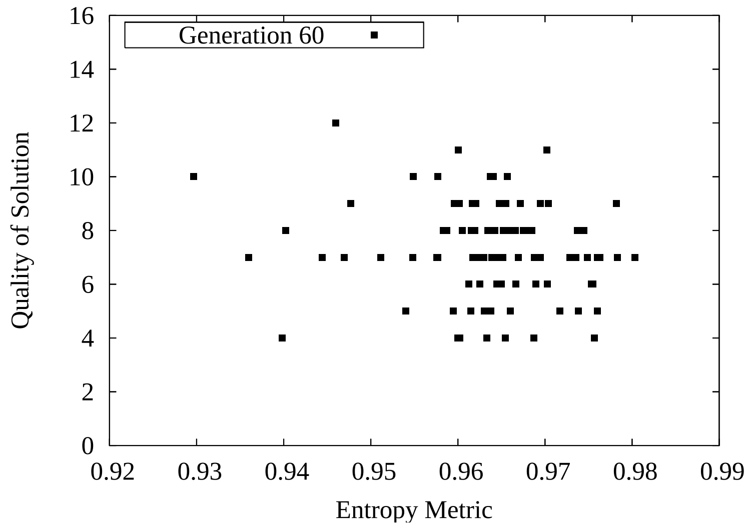


Figure 7.63: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 60. **90** Runs.

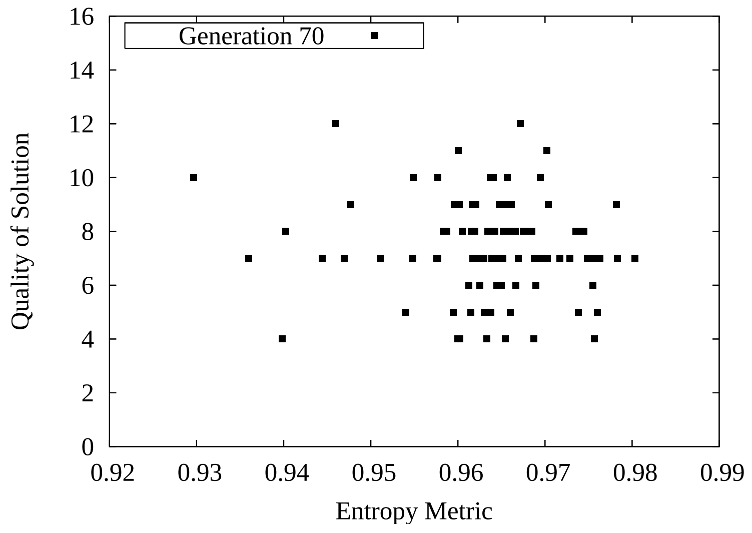


Figure 7.64: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 70. **90** Runs.

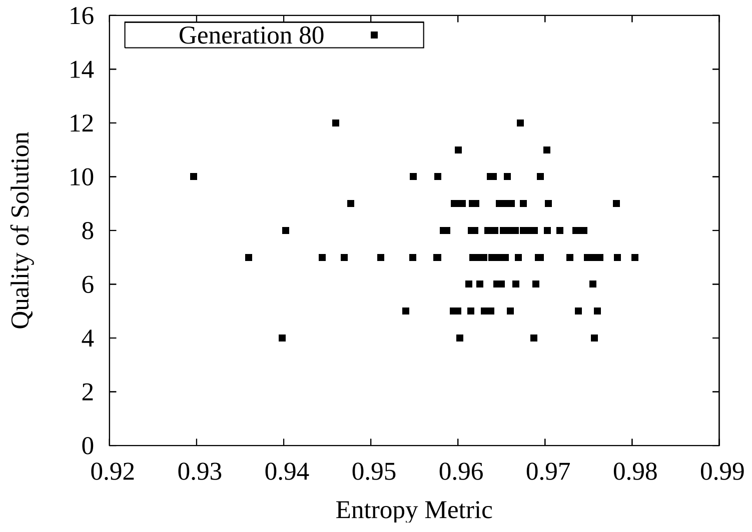


Figure 7.65: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 80. **90** Runs.

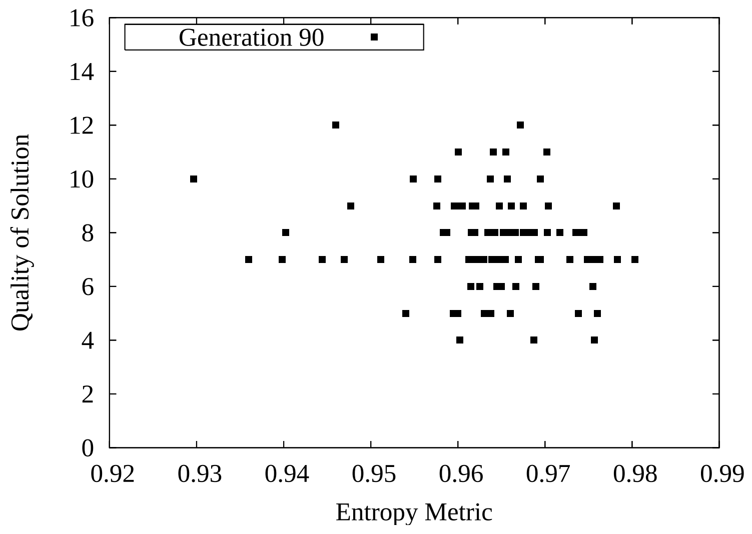


Figure 7.66: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 90. **90** Runs.

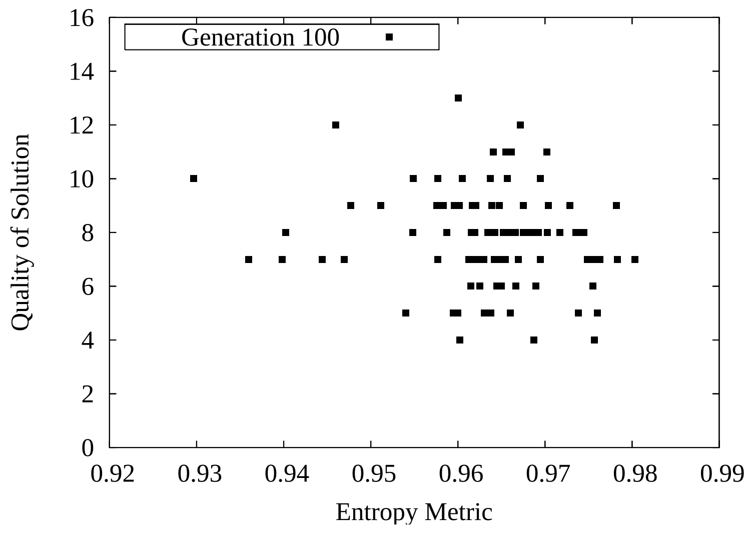


Figure 7.67: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 100. **90** Runs.

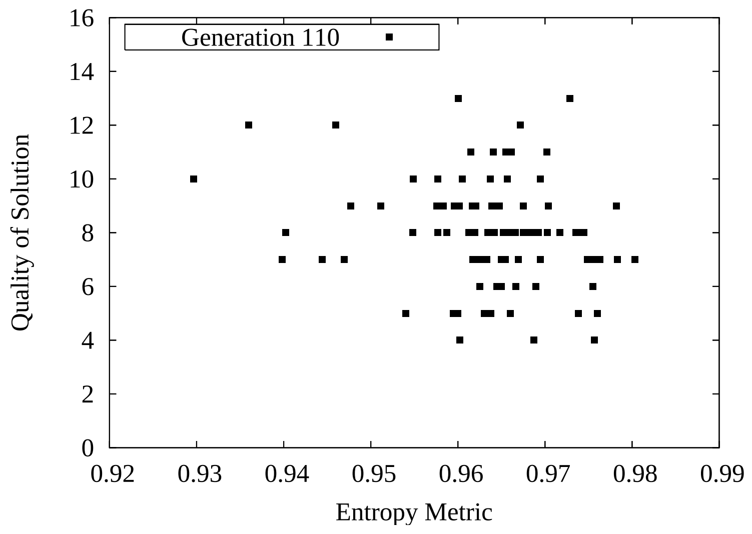


Figure 7.68: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 110. **90** Runs.

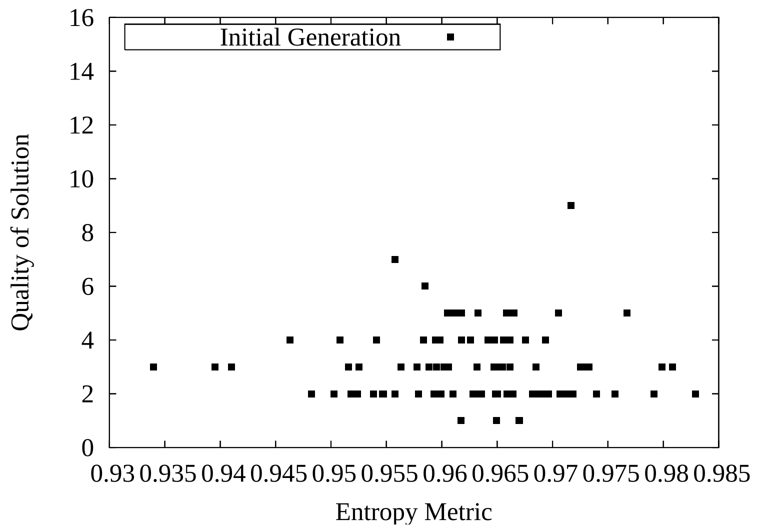


Figure 7.69: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at initial generation. **90** Runs.

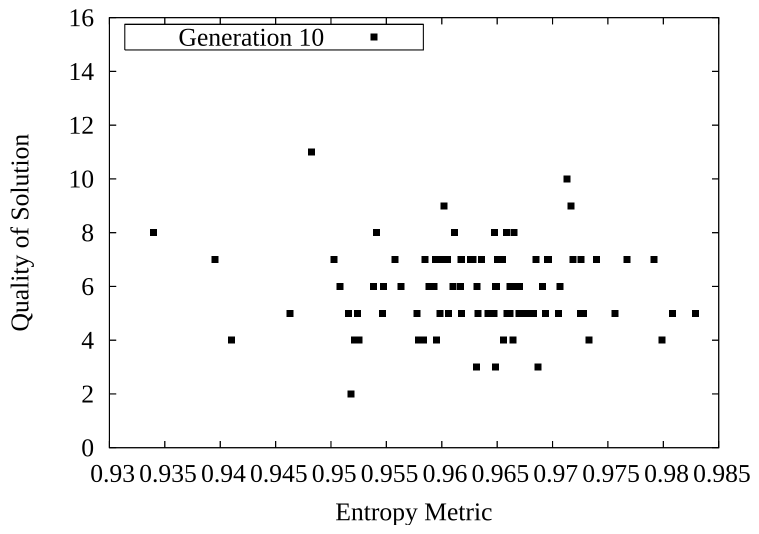


Figure 7.70: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 10. $\mathbf{90}$ Runs.

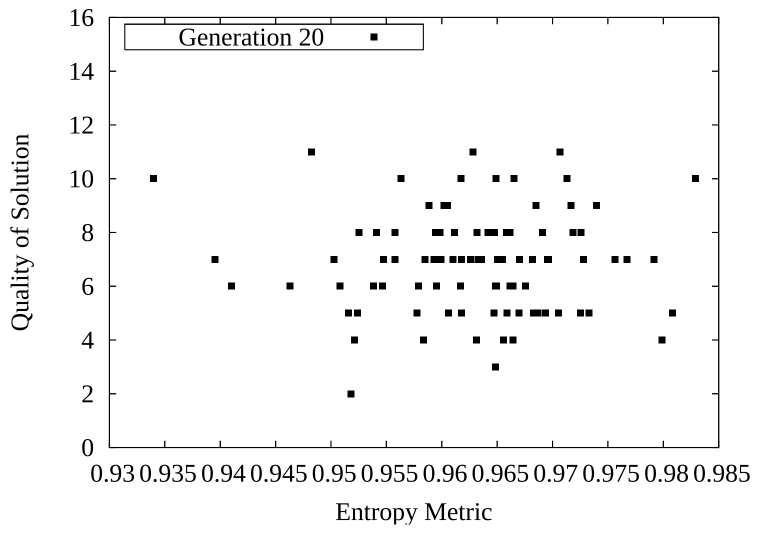


Figure 7.71: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 20. $\mathbf{90}$ Runs.

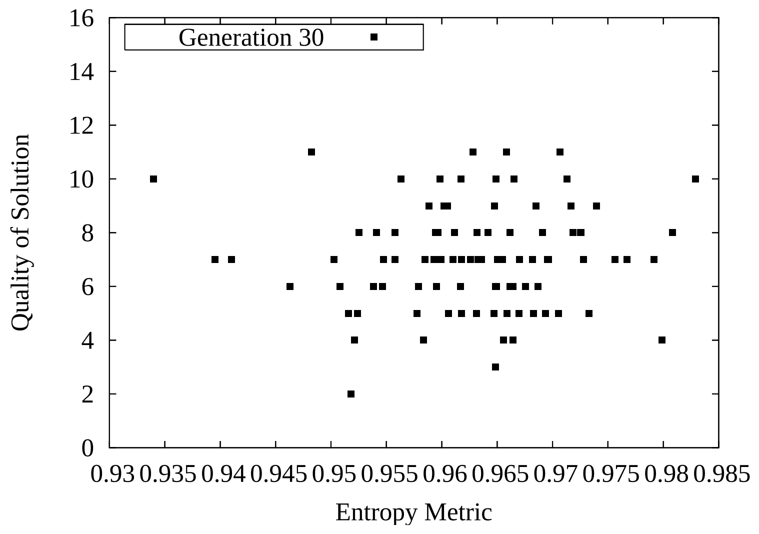


Figure 7.72: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 30. **90** Runs.

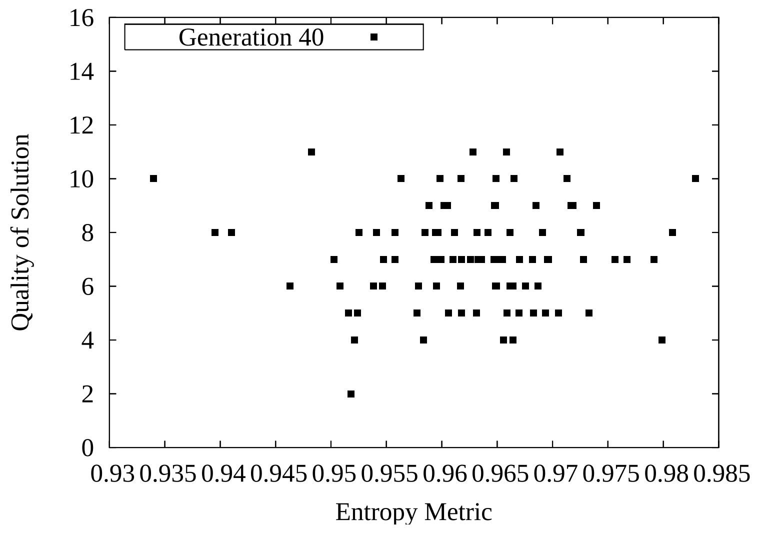


Figure 7.73: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 40. **90** Runs.

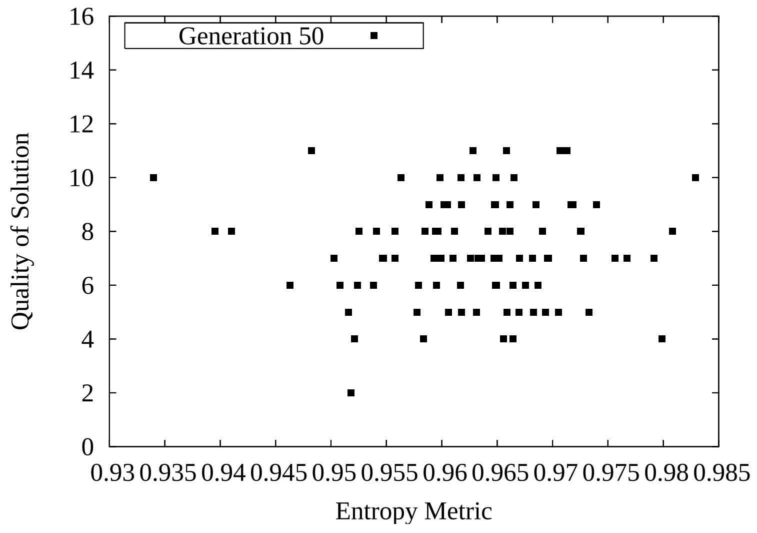


Figure 7.74: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 50. **90** Runs.

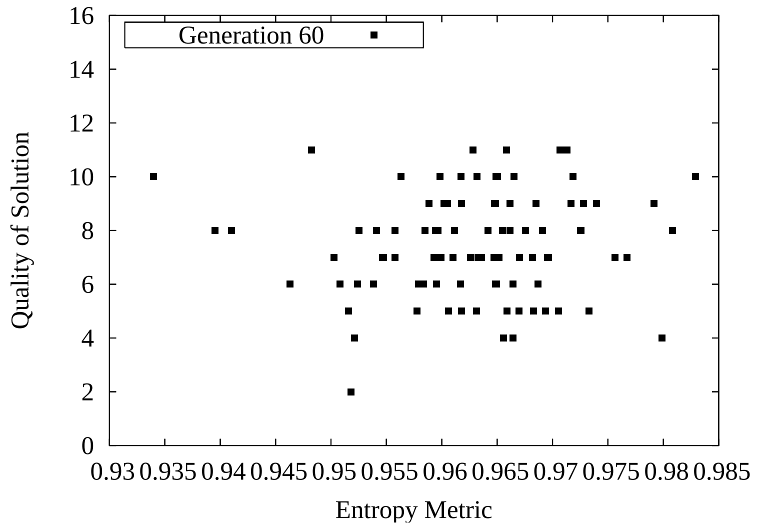


Figure 7.75: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 60. **90** Runs.

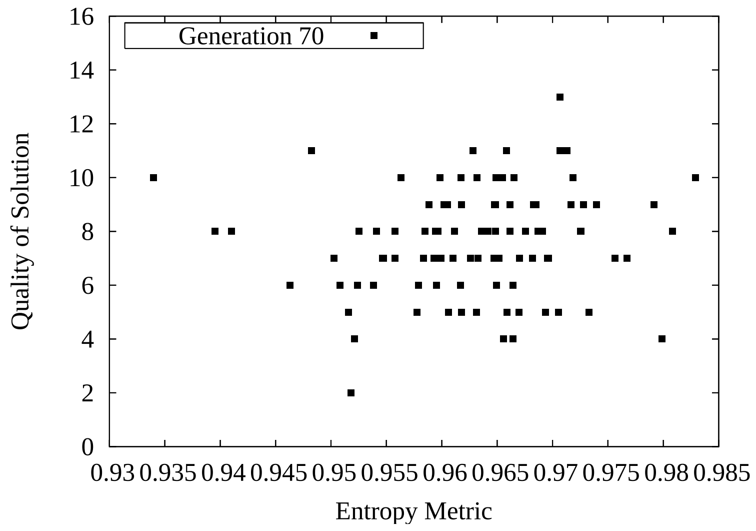


Figure 7.76: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 70. **90** Runs.

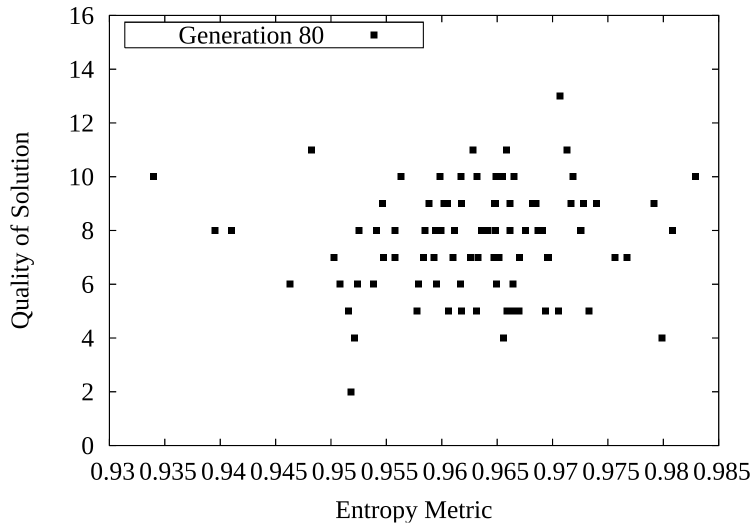


Figure 7.77: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 80. **90** Runs.

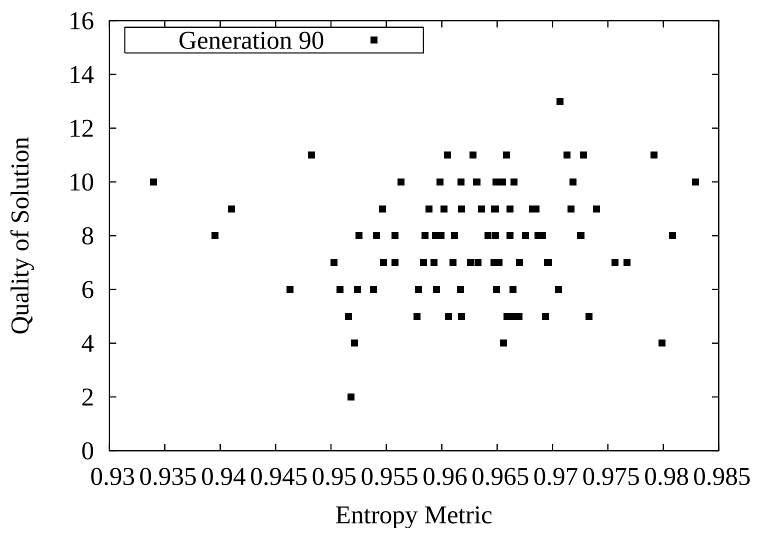


Figure 7.78: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 90. **90** Runs.

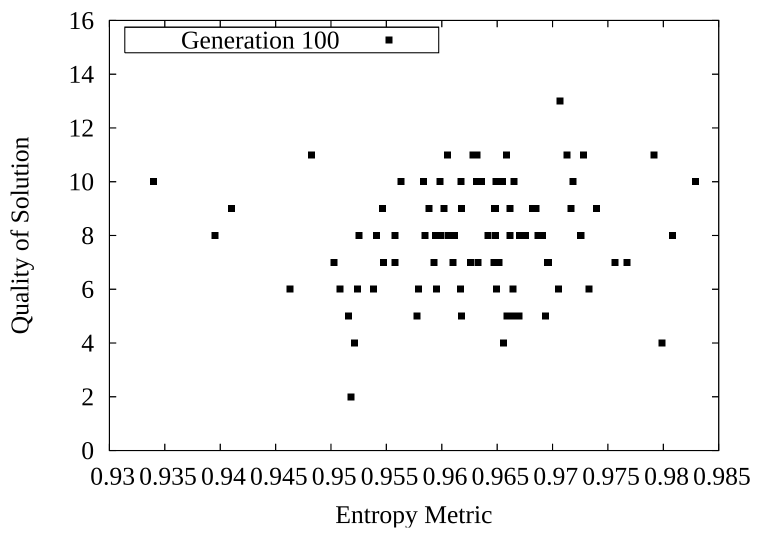


Figure 7.79: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 100. **90** Runs.

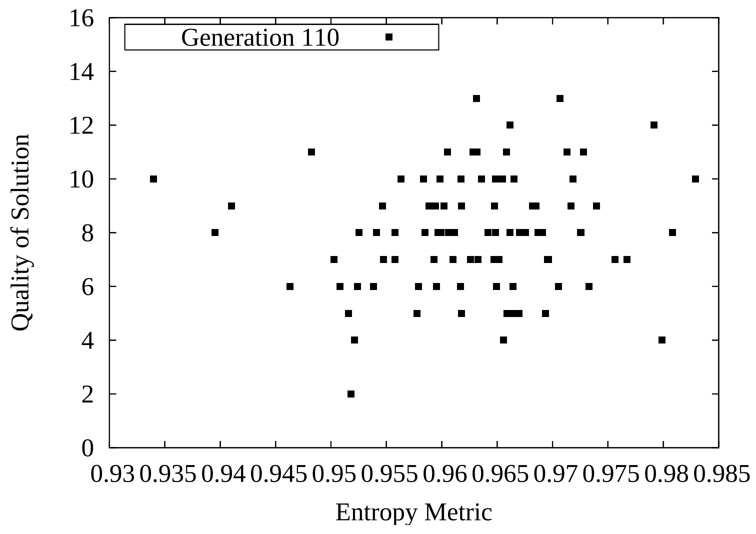


Figure 7.80: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 110. $\mathbf{90}$ Runs.

7.1.6 Results for Specific Hypothesis 1.6

The sixth specific hypothesis to test is the following: Equation 7.1 is satisfied for the snake-in-the-box problem using the entropy metric to measure diversity, number of generations to reach a global maximum to measure expected performance, and parameter set 1.2.

Figure 7.81 shows the number of generations to reach a global maximum for the corresponding seeds.[1] No statistically significant correlation between the entropy metric and the expected number of generations to reach a global maximum was found. The Pearson and Fisher's coefficient values obtained were 0.123 and 1.153.

In order to see if the seeds that converge (17 points as in Figure 7.81) and the ones that do not converge, are correlated, the logistic regression test was applied. This gives a p value of 0.0075 for the best Ξ^2 model value, i.e., the model cannot adequately fit the model. Therefore, there is no correlation between the two sets.

[1]There were seeds that did not reach a global maximum in the $100,000$ generations.

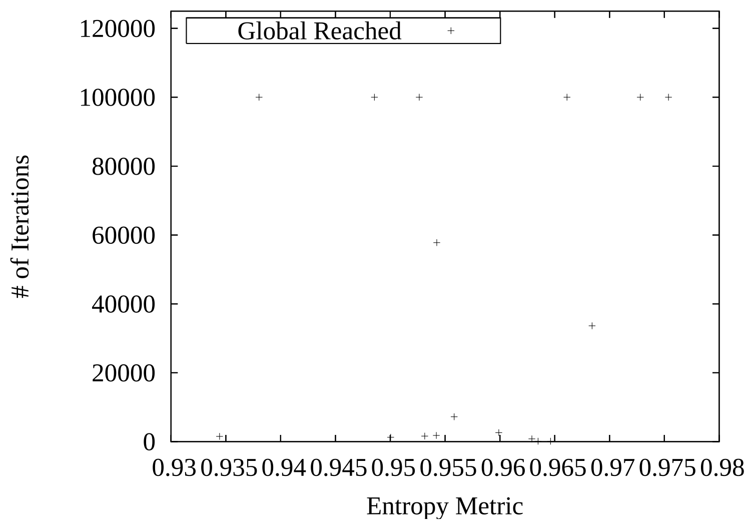


Figure 7.81: Entropy metric vs. number of generations to reach the global maximum. $100,000$ Generations. Snake-in-the-box Problem. **90** Runs.

7.1.7 Results for Specific Hypothesis 1.7

The seventh specific hypothesis to test is the following: Equation 7.1 is satisfied for the snake-in-the-box problem using the Hamming metric to measure diversity, solution quality to measure expected performance, and parameter set 1.2.

Snapshots were taken at generations at each 10 generations until generation 110 as is shown from Figure 7.82 to 7.93. No statistically significant correlation between the entropy metric and the expected quality of the solution was discovered as the Pearson and Fisher's coefficient show in Tables 7.8 and 7.9 that correspond to the snapshots as in Figures 7.82 to 7.117.

		Generation					
Trial		0	10	20	30	40	50
1	R_{XY}	−0.068	−0.011	−0.005	−0.015	−0.020	−0.013
2	R_{XY}	0.187	−0.116	−0.119	−0.191	−0.148	−0.156
3	R_{XY}	−0.013	−0.049	−0.005	0.044	0.027	0.026
		Generation					
Trial		60	70	80	90	100	110
1	R_{XY}	−0.037	−0.037	−0.009	0.007	0.017	0.008
2	R_{XY}	−0.160	−0.114	−0.103	−0.160	−0.165	−0.230
3	R_{XY}	0.070	0.112	0.100	0.119	0.116	0.124

Table 7.8: Pearson's Coefficients for the Snake-in-the-box problem. Hamming Metric. No linear correlation found.

		Generation											
Trial		0	10	20	30	40	50	60	70	80	90	100	110
1	Z	−0.64	−0.10	−0.05	−0.14	−0.19	−0.12	−0.34	−0.34	−0.08	0.06	0.16	0.07
2	Z	1.76	−1.09	−1.11	−1.80	−1.39	−1.47	−1.50	−1.07	−0.96	−1.50	−1.55	−2.18
3	Z	−0.12	−0.46	−0.048	0.41	0.25	0.24	0.65	1.05	0.94	1.11	1.09	1.16

Table 7.9: Fisher's Coefficients for the Snake-in-the-box problem. Hamming Metric. No linear correlation found.

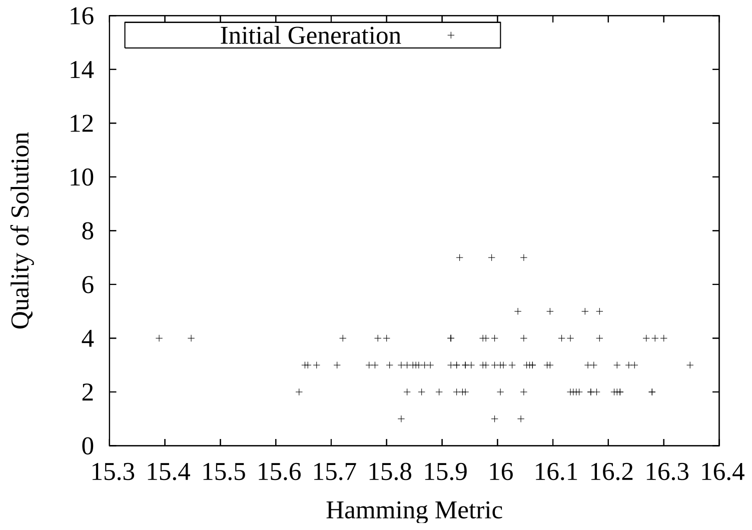


Figure 7.82: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at initial generation. $\mathbf{90}$ Runs.

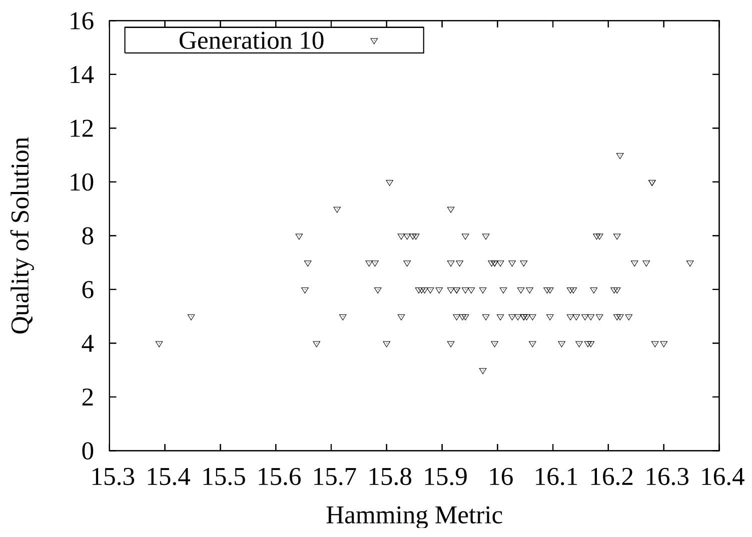


Figure 7.83: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 10. **90** Runs.

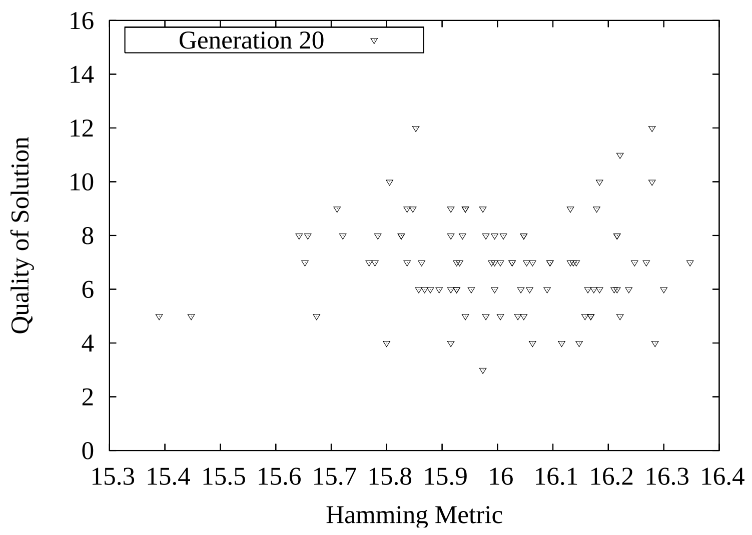


Figure 7.84: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 20. **90** Runs.

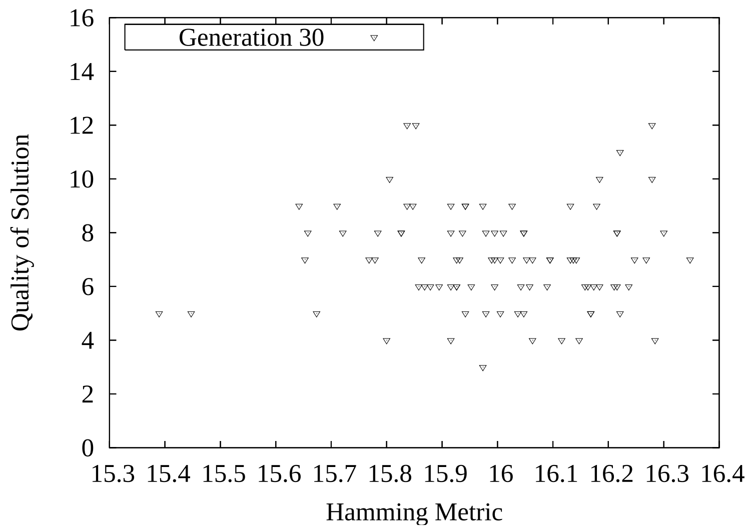


Figure 7.85: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 30. **90** Runs.

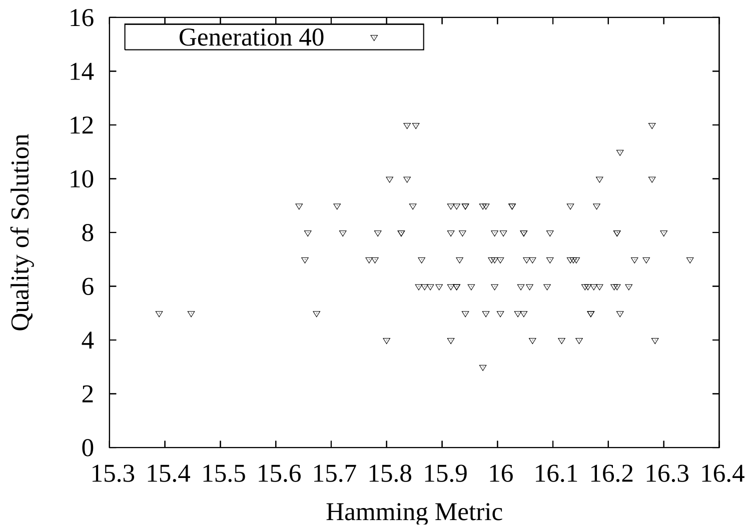


Figure 7.86: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 40. **90** Runs.

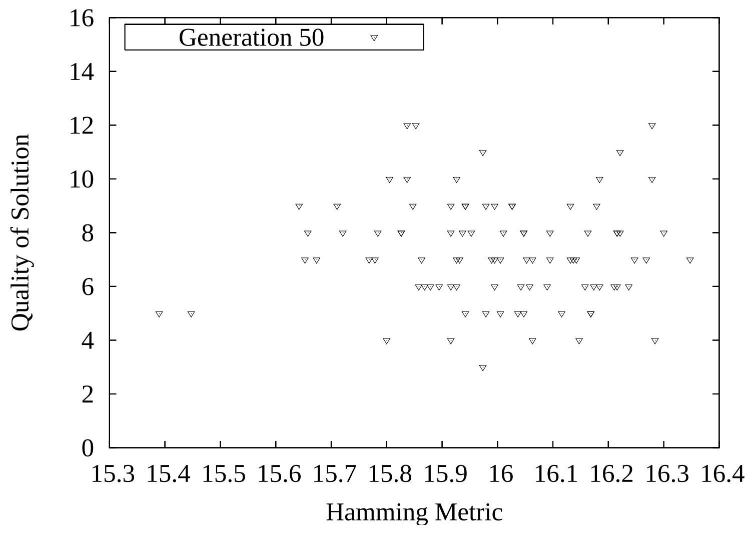


Figure 7.87: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 50. **90** Runs.

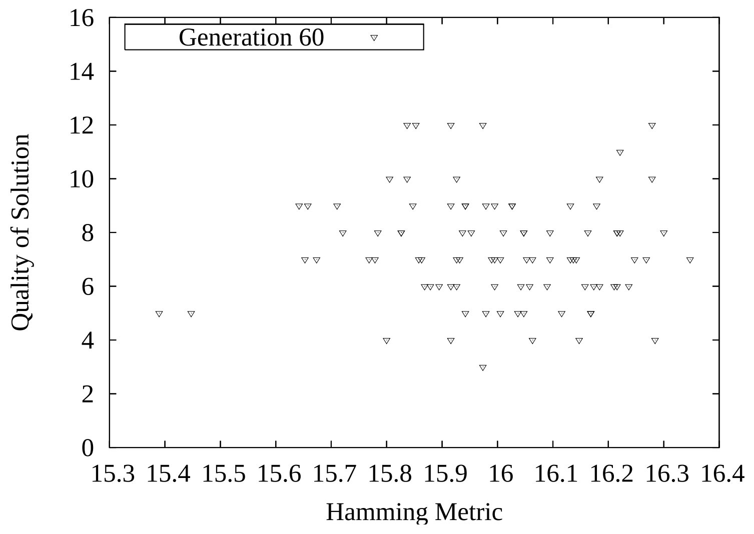


Figure 7.88: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 60. **90** Runs.

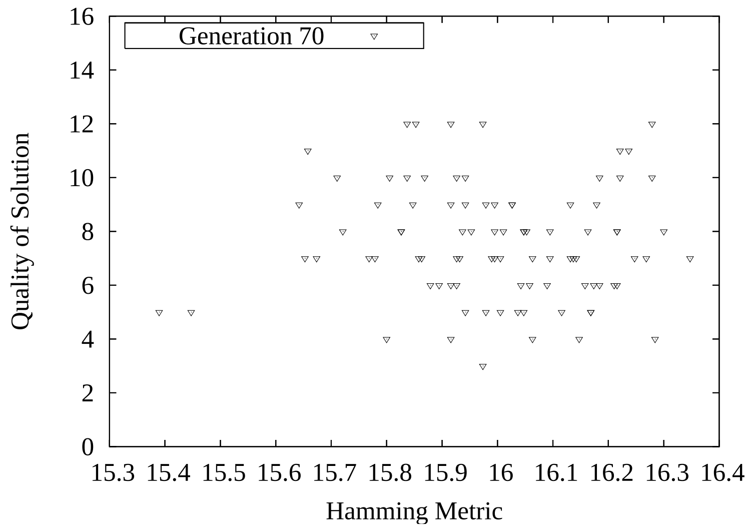


Figure 7.89: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 70. **90** Runs.

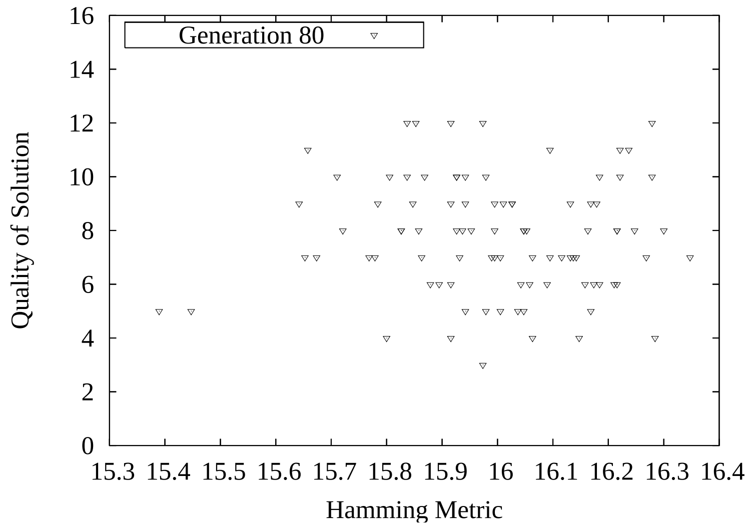


Figure 7.90: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 80. **90** Runs.

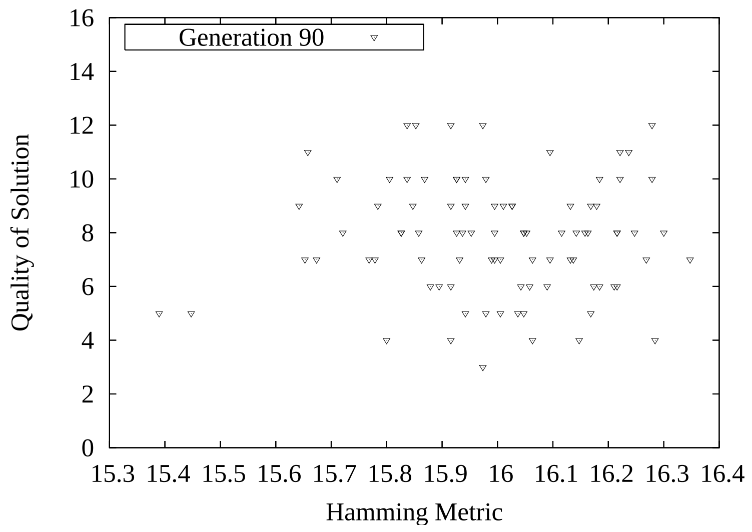


Figure 7.91: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 90. **90** Runs.

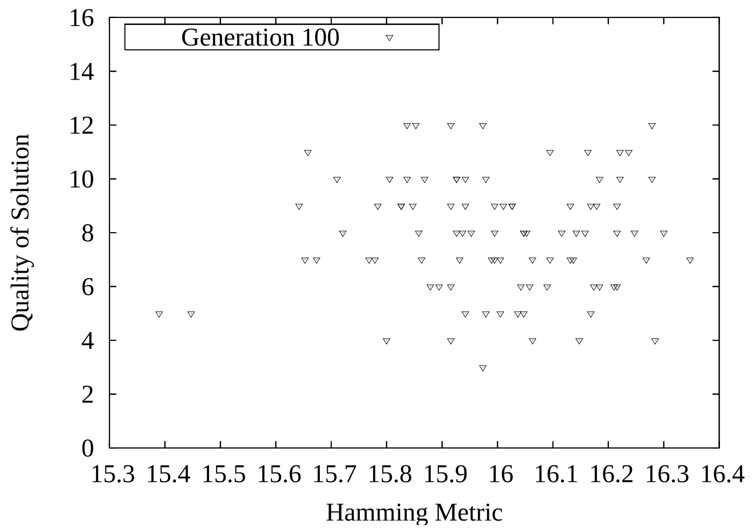


Figure 7.92: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 100. **90** Runs.

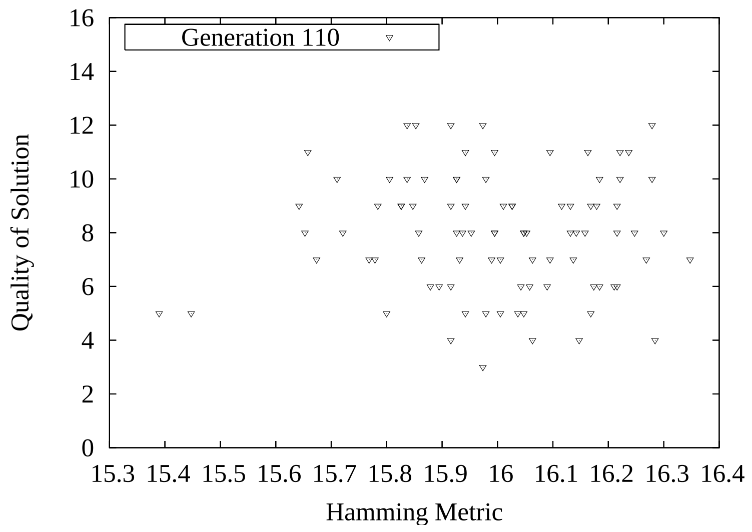


Figure 7.93: Hamming metric vs. quality of the solution. Snake-in-the-box Problem. Trial 1. Snapshot at generation 110. **90** Runs.

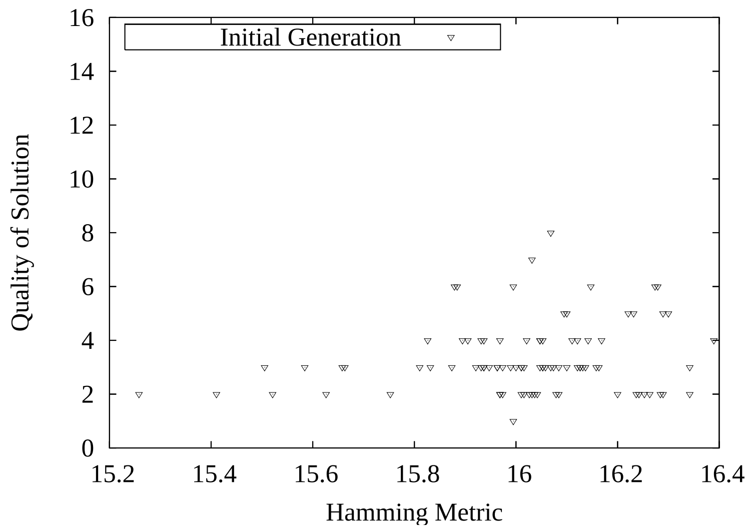


Figure 7.94: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at initial generation. **90** Runs.

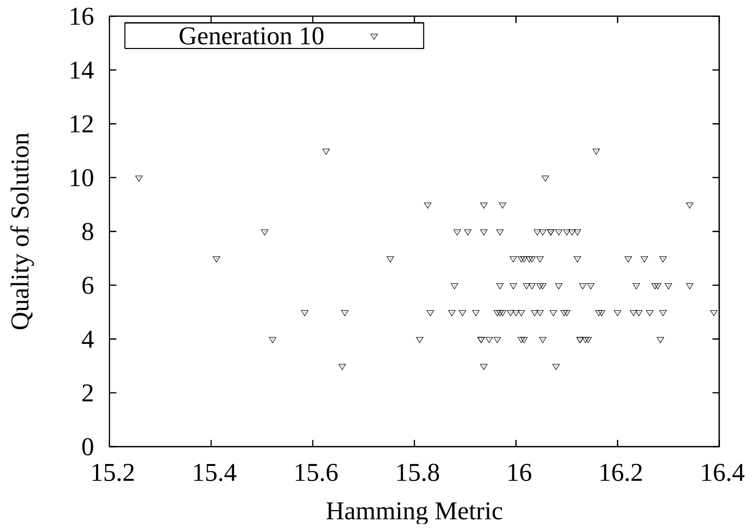


Figure 7.95: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 10. **90** Runs.

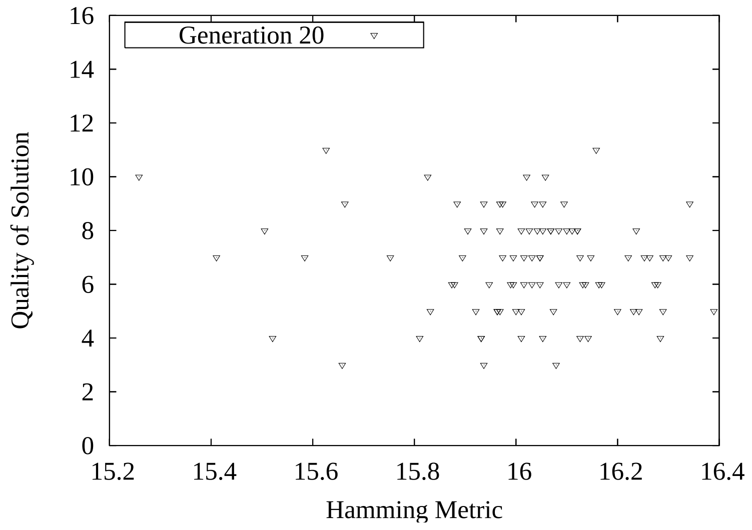


Figure 7.96: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 20. **90** Runs.

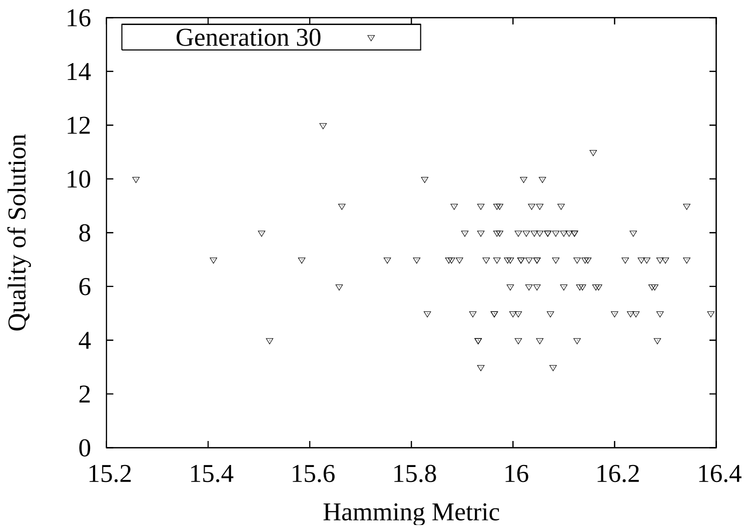


Figure 7.97: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 30. **90** Runs.

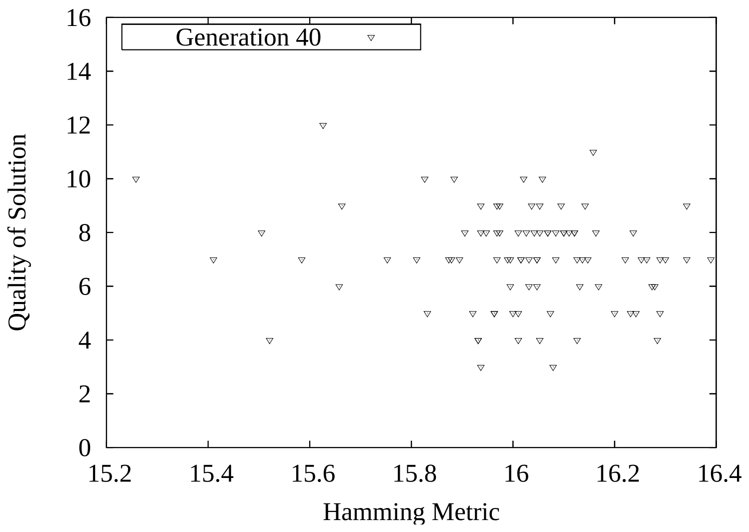


Figure 7.98: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 40. **90** Runs.

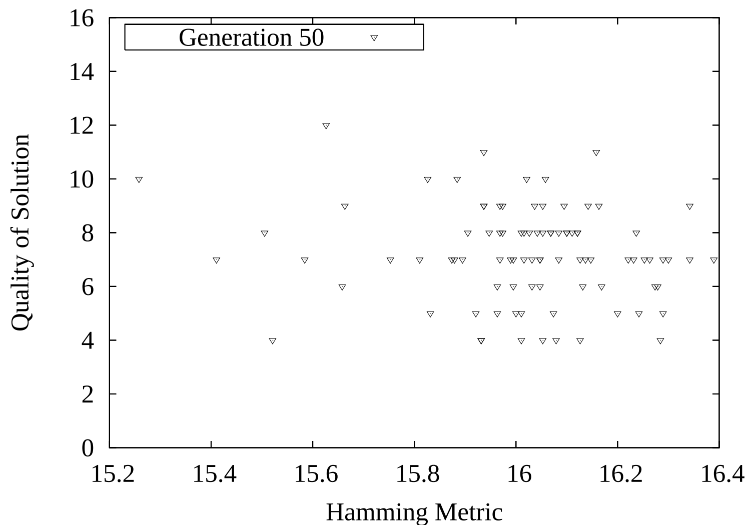


Figure 7.99: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 50. **90** Runs.

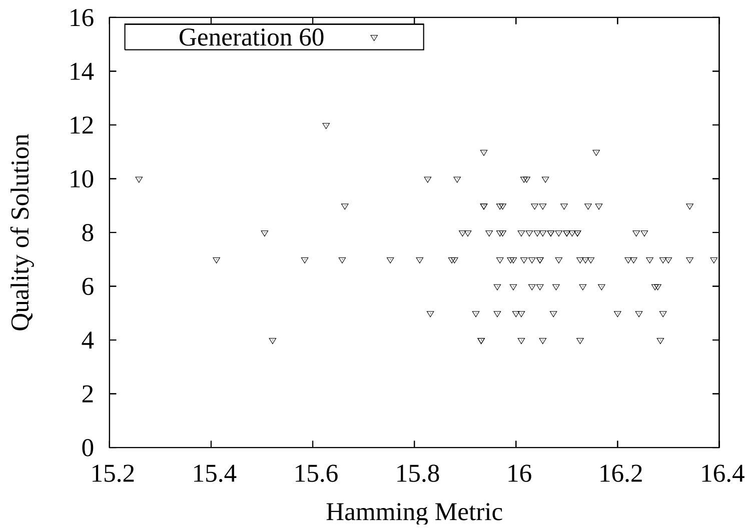


Figure 7.100: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 60. **90** Runs.

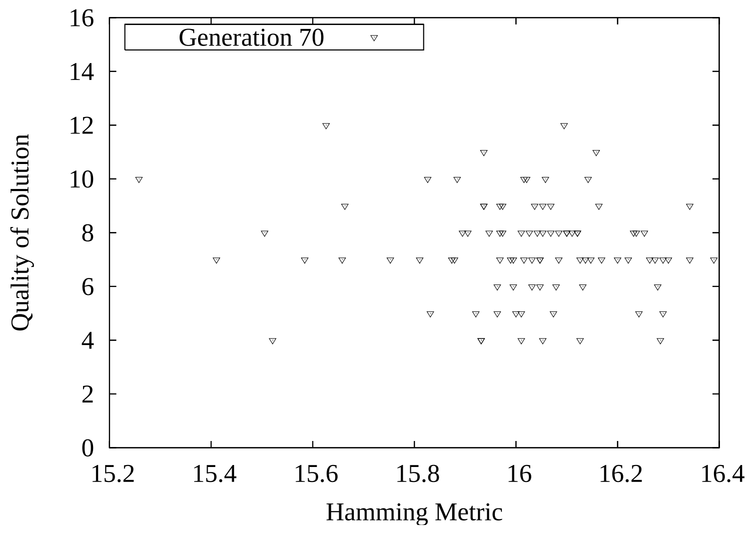


Figure 7.101: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 70. **90** Runs.

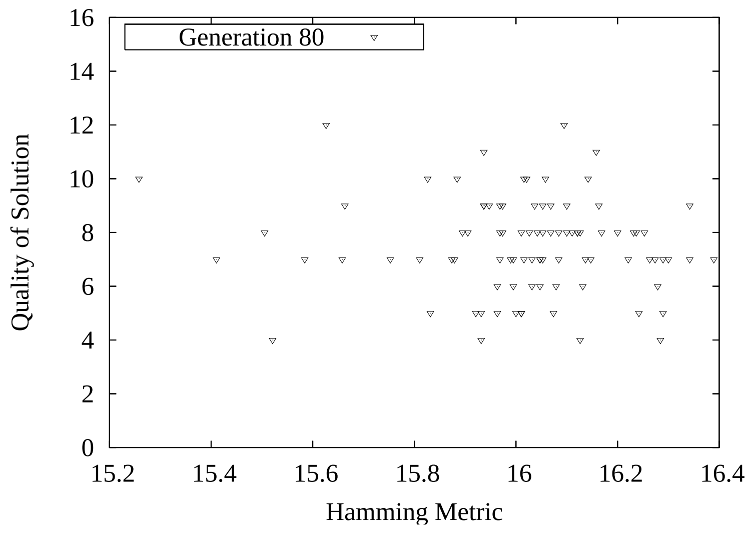


Figure 7.102: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 80. **90** Runs.

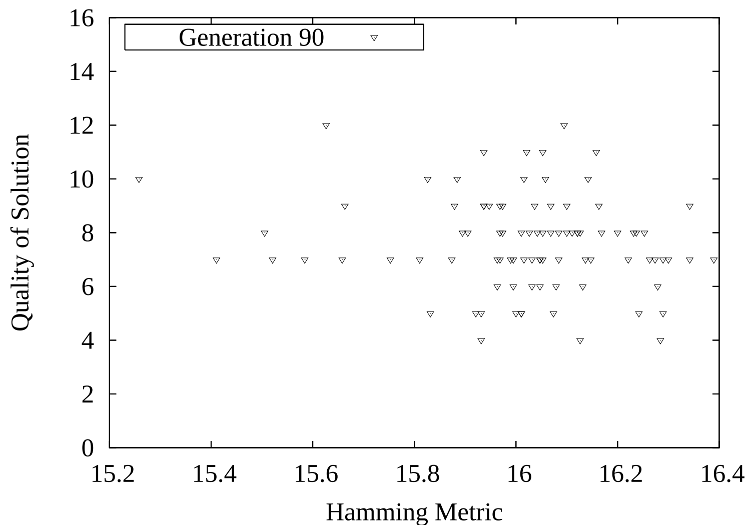


Figure 7.103: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 90. **90** Runs.

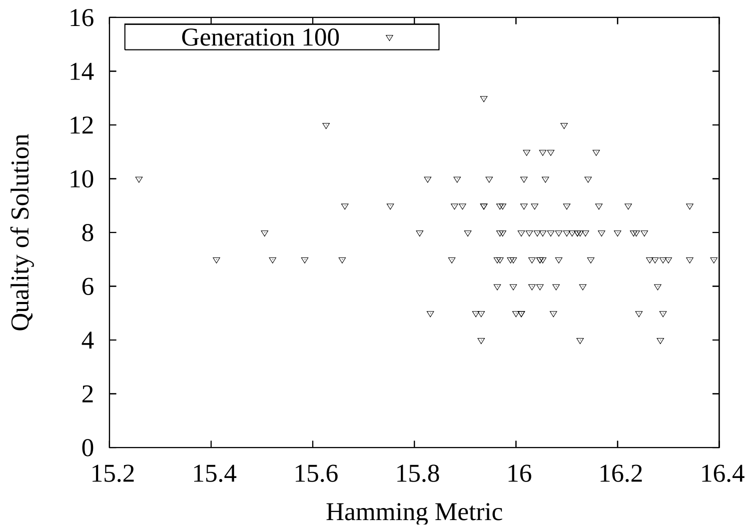


Figure 7.104: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 100. **90** Runs.

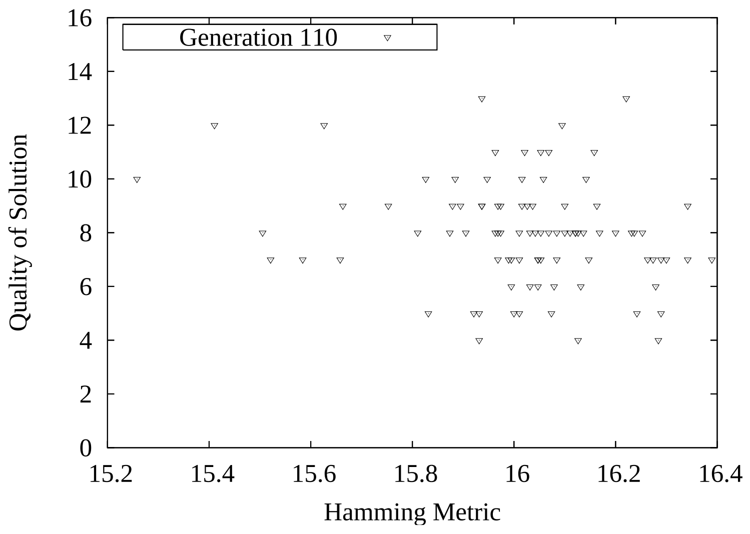


Figure 7.105: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 2. Snapshot at generation 110. 90 Runs.

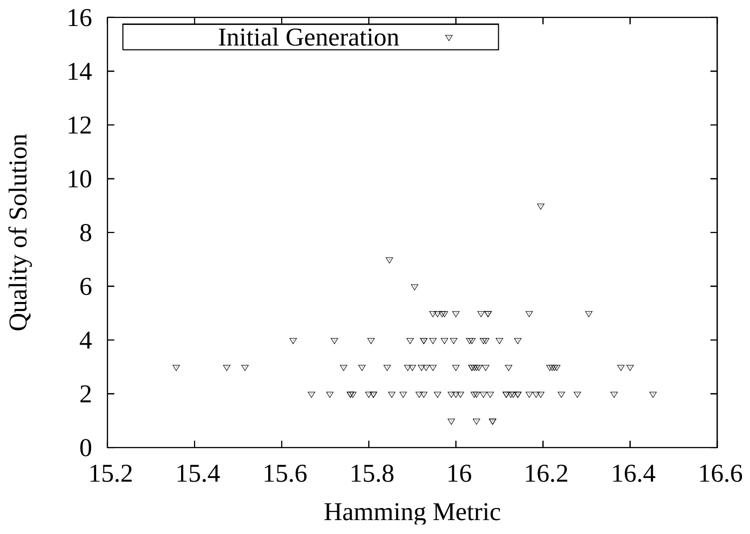


Figure 7.106: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at initial generation. 90 Runs.

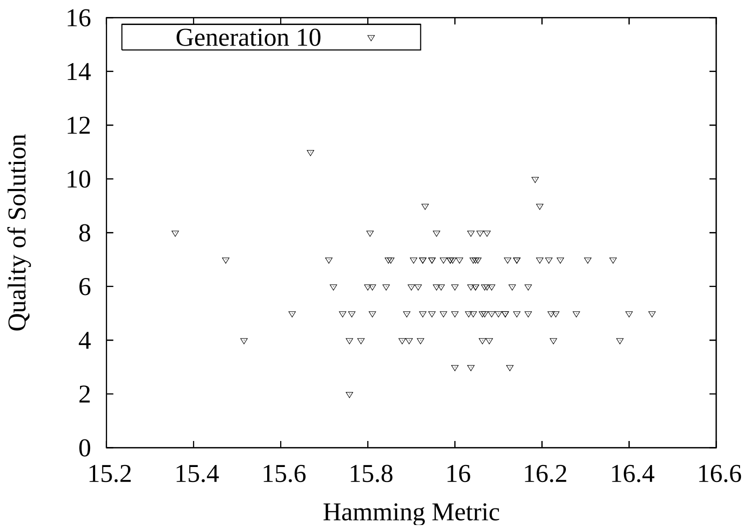


Figure 7.107: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 10. **90** Runs.

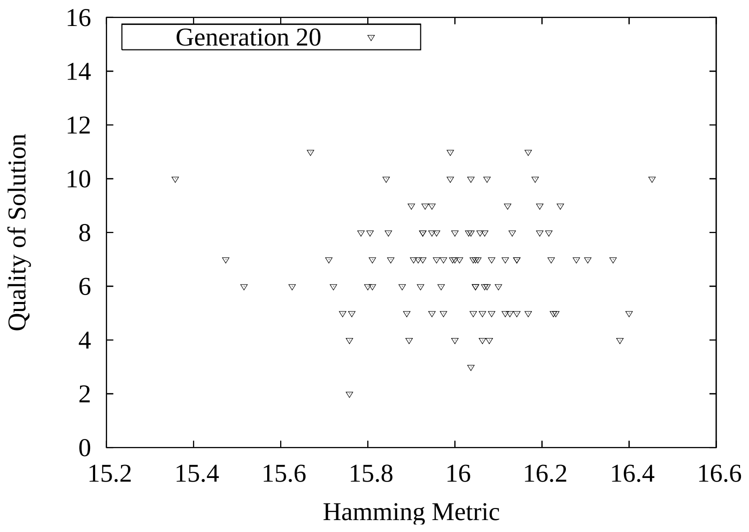


Figure 7.108: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 20. **90** Runs.

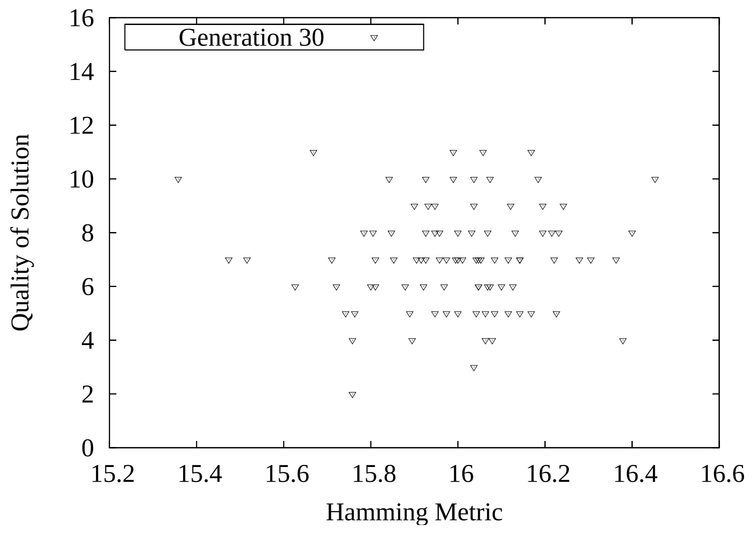


Figure 7.109: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 30. **90** Runs.

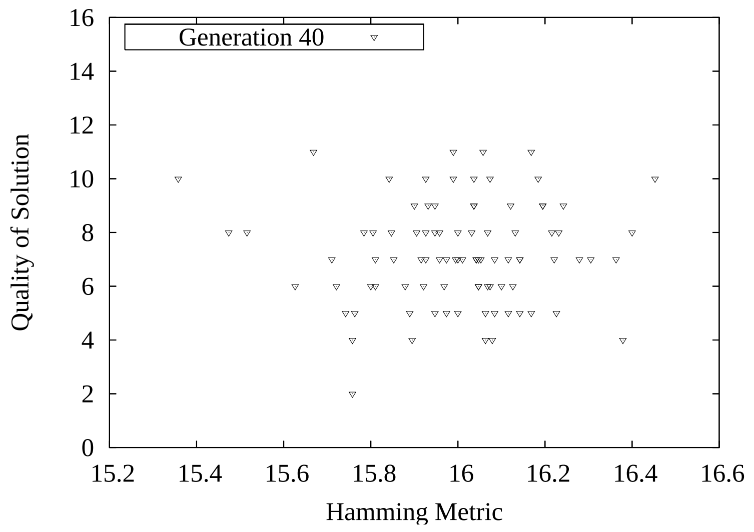


Figure 7.110: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 40. **90** Runs.

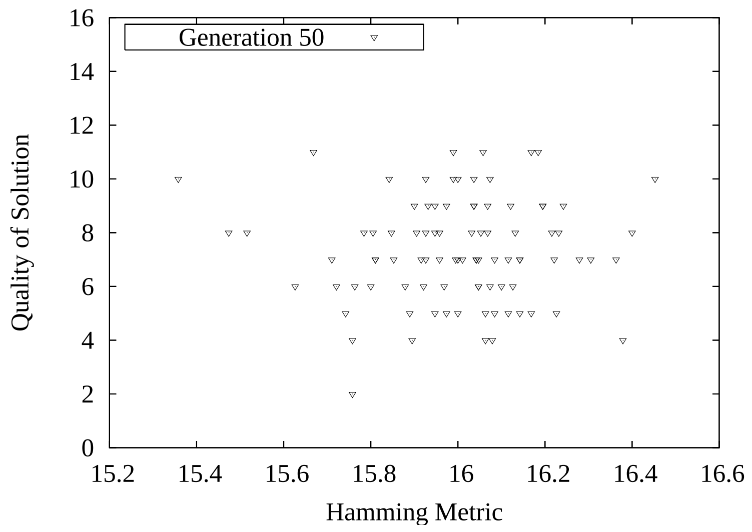


Figure 7.111: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 50. **90** Runs.

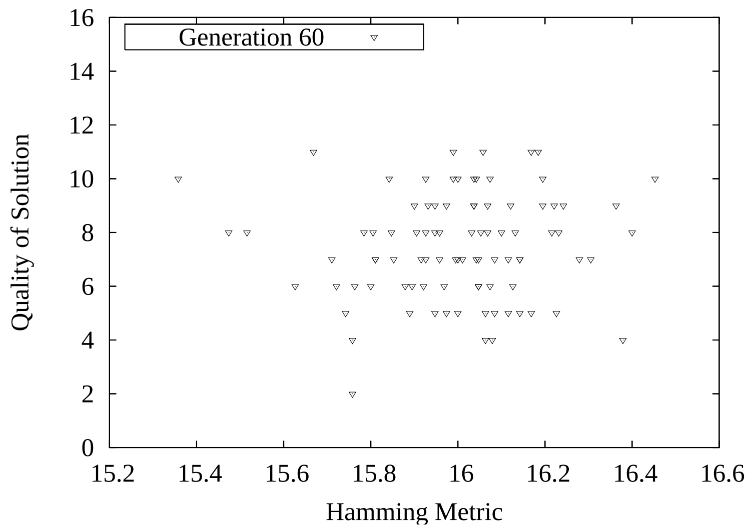


Figure 7.112: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 60. **90** Runs.

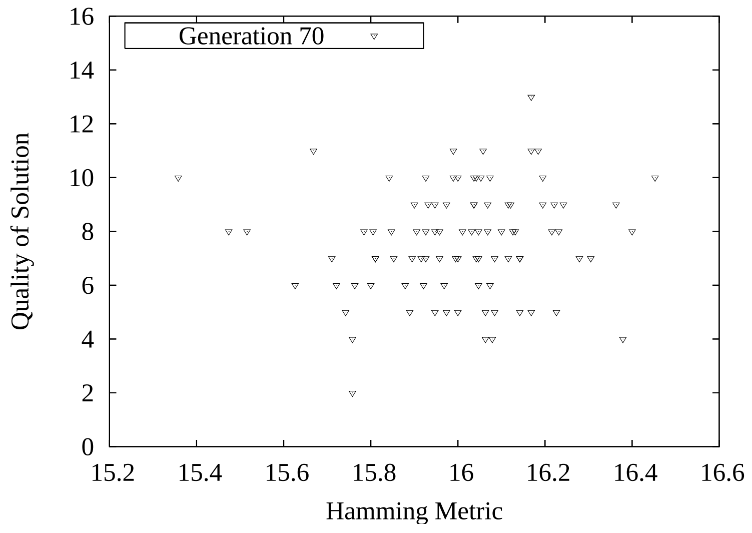


Figure 7.113: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 70. **90** Runs.

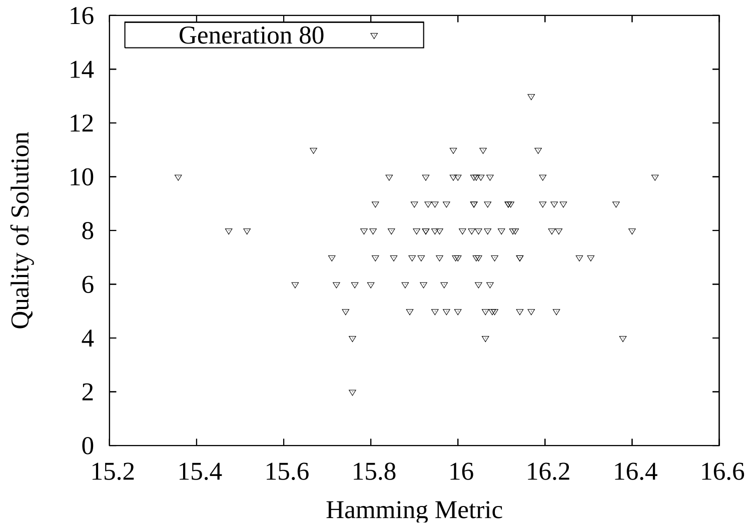


Figure 7.114: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 80. **90** Runs.

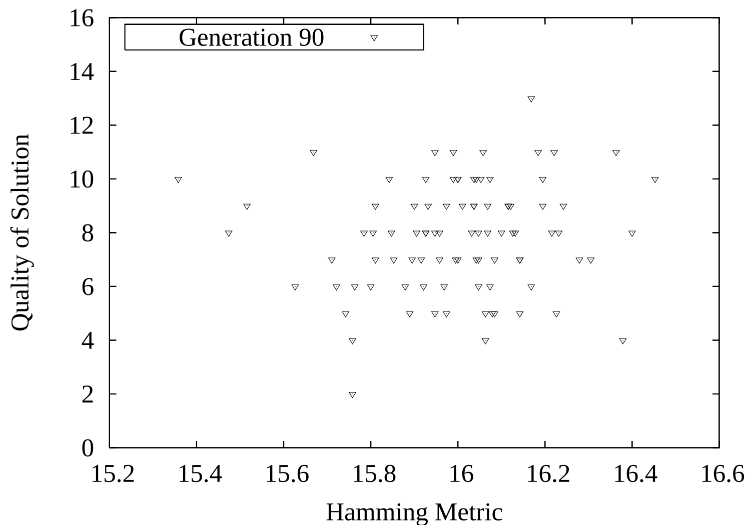


Figure 7.115: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 90. **90** Runs.

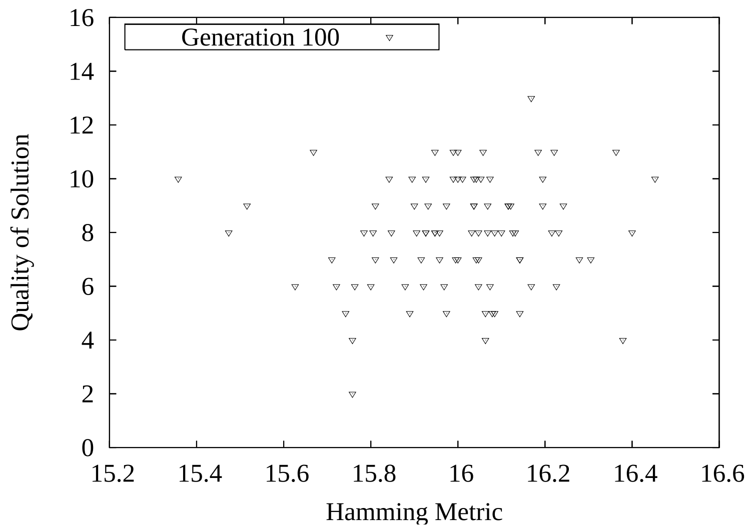


Figure 7.116: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 100. **90** Runs.

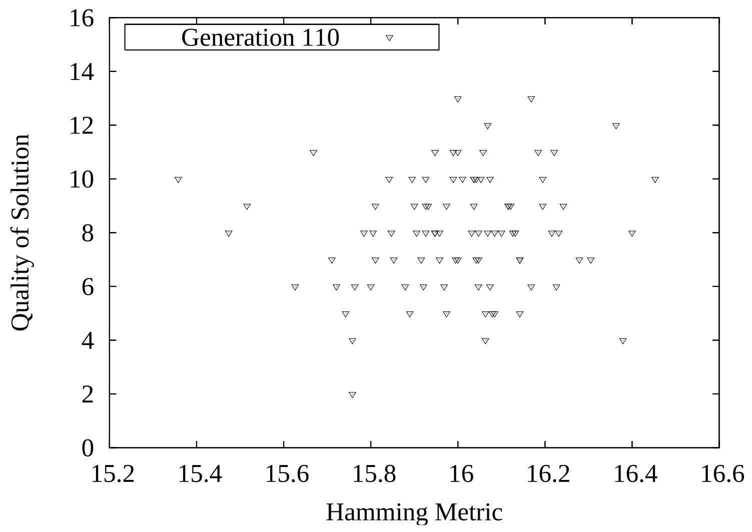


Figure 7.117: Entropy vs. quality of the solution. Snake-in-the-box Problem. Trial 3. Snapshot at generation 110. **90** Runs.

7.1.8 Results for Specific Hypothesis 1.8

The eighth specific hypothesis to test is the following: Equation 7.1 is satisfied for the snake-in-the-box problem using the Hamming metric to measure diversity, number of generations to reach a global maximum to measure expected performance, and parameter set 1.2.

Figure 7.118 takes into account the number of generations to reach a global maximum for the corresponding seeds.[2] No statistically significant correlation between the Hamming metric and the expected number of generations to reach a global maximum was discovered as the Pearson and Fisher's coefficient values of 0.125 and 1.172 were obtained (see Figure 7.118 for the snapshot at last generation $(100,000)$).

In order to see if the seeds that converge (17 points as in Figure 7.118) and the ones that do not converge, are correlated, the logistic regression test was applied. This gives a p value of 0.0039 for the best Ξ^2 model value, i.e., the model cannot adequately fit the model. Therefore, there is no correlation between the two sets.

[2]There were seeds that did not reach a global maximum in the $100,000$ generations.

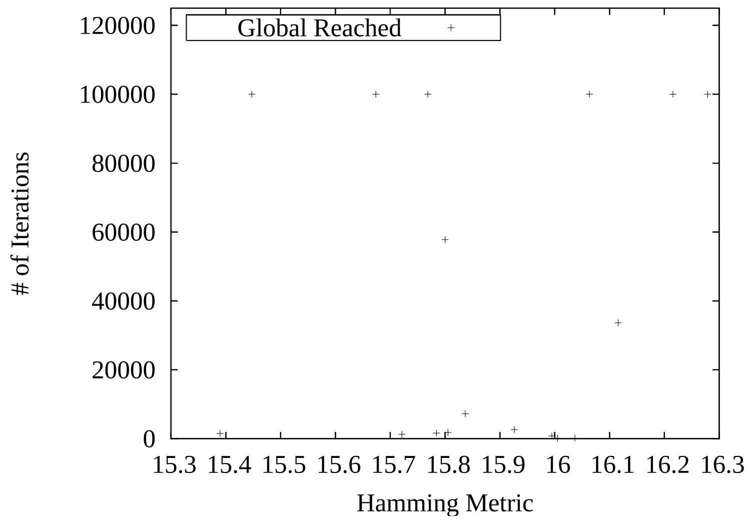


Figure 7.118: Hamming metric vs. number of generations to reach the global maximum. $100,000$ Generations. Snake-in-the-box Problem. **90** Runs.

7.2 Internal vs. External Parameters

A fitness function is needed for a GA to work, and it appears natural that the combination of objectives and constraints into a single scalar function using arithmetic operations is appropriate (Coello, 1998; Mé, 1998). One problem with this approach, however, is that accurate scalar information must be provided on the range of objectives and constraints, to avoid one of them from dominating the other (Coello, 1998). One possible solution, then, is to try to join the objectives with the constraints with *internal parameters*, i.e., information that belongs to the problem itself, thereby avoiding external tuning. The building of the fitness function is so complex that, using internal or external parameters, any optimal point obtained will be a function of the coefficients used to combine objectives and constraints (Coello, 1998). However, it is possible that with internal parameters, the averages of false positive and/or false negative may decrease.

Equation 7.3 is normalized, this means that it is good for finding local maxima, while at the same time a false negative ratio can be expected depending on the sizes of the chromosome and the population (Diaz-Gomez and Hougen, 2007e,d). For the case of the intrusion detection problem, we propose the use of the *union operator* which is a mechanism that stores the first local minimum found (if that is the case, because the file could have no intrusions at all) and thereafter, if another local is found, it performs the union of the previous intrusions with new ones and, if there is no violations of constraints, then the new intrusion is added to the solution set. In this way it avoids a high false negative ratio.

Equation 7.4 requires external tunning: α and β should be such that it gives no false positives nor false negatives. However, this is a difficult task because when we try to avoid false positives (let say increasing β), then it is possible that false negatives appear and viceversa (see Section 7.2.1). So, if we are going to compare Equation 7.4 with Equation 7.3, and 7.4 has been tunned to avoid false negatives (or false positives), then, the results could be ambiguous in the sense that there could be a statistically difference between the two for the false positive (or false negative) case, but not for the false negative (or positive) case, where the algorithm was trained. To avoid this, we need a score that takes into account both, the false positives and false negatives. The Hanssen and Kuipers' score, is considered a good predictor, and it takes into account not only the false positives (B) and false negatives (C) but the correct positives (A) and correct negatives (D). The formula used for the Hanssen and Kuipers' score is $Skill = A/(A + C) - B/(B + D)$ which tell us how well the algorithm, with the corresponding fitness function, separates the "intrusion" from the "non-intrusion". The range of the Hanssen and Kuipers' score is between -1 to 1, being 1 perfect score, and 0 an indication of no skill.[3]

Research Question 2:

The parameters that join the objective and the constraint in the function that evaluates the structure A: Do internal parameters combined with the union operator result in better performance than external parameters (i.e., exogenous parameters that may be tuned according to the input)?

General Form of Hypothesis 2.

If G_{F_i} is a GA with a fitness function with internal parameters which uses the union operator and G_{F_e} is a GA with a fitness function with external parameters, then

$$X(P, G_{F_i}) \geq X(P, G_{F_e}) \tag{7.2}$$

where $X(P, G_F)$ is the Hanssen and Kuipers' score of the GA with fitness function F on population P.

[3]Thanks to Dr. Michael Richman for suggesting this score.

Parameter set 2: Population size $N = 50$, chromosome length $l = 24$, 2-tournament selection ($75\% - 25\%$ which means that two individuals are selected randomly and with a probability of 75% the best is chosen, and with a probability of 25% the worst is chosen), probability of 1-point crossover $p_c = 0.6$, probability of mutation $p_m = 0.0083$ per bit, stop criterion is $1,000$ generations.

Parameter Justification: This set of parameters was used by Mé (1993), except the 2-tournament selection pressure ($75\% - 25\%$), which is not defined in Mé (1993), but that we used in previous tests with good results (Diaz-Gomez and Hougen, 2006a, 2007e).

Fitness Functions to use: Fitness function with internal parameter suggested by Diaz-Gomez and Hougen (2006a) (see Section 5.1)

$$F(I) = \frac{\sum_{j=1}^{N_e}(AE * I)_j - \sum_{j=1}^{N_e} max[0, (AE * I)_j - OV_j]}{\sum_{j=1}^{N_e}(AE * I)_j} \tag{7.3}$$

Fitness function with external parameter suggested by Mé (1998) (see Section 3.3)

$$F(I) = \alpha + \sum_{i=1}^{N_a} W_i * I_i - \beta * T^2. \tag{7.4}$$

Specific Hypothesis 2.1

Equation 7.2 is satisfied for the off-line intrusion detection problem, using as fitness function F_i Equation 7.3 with internal parameters and the union operator, and as fitness function F_e Equation 7.4. Parameter set to use 2. Averages of false positive and false negative are measured in order to obtain the Hassen and Kuipers' score.

Method

Use 5 different sets of external parameters (α, β) for fitness function 7.4 which are: $(392.0, 1.0)$, $(4.0, 0.05)$, $(50.0, 7.0)$, $(50.0, 1.0)$, $(50.0, 0.5)$.[4] Run 30 times for each set of parameters finding the average number of false positives and negatives in the 30 runs, in order to obtain the best set of parameters in terms of the Hansen and Kuipers' score.

[4]Mé (1993) used $\alpha = 50$ and $\beta = 1.0$, so we begin from there to test if a better set of parameters could be obtained,as it happened with parameters $\alpha = 392.0$ and $\beta = 1.0$.

Equation 7.4 will be compared with equation 7.3 using the best set of external parameters found in previous step for Equation 7.4. Run the GA 30 times with the fitness function as in Equation 7.3 and with the union operator, and 30 times with Equation 7.4, using 25 different scenarios (from no intrusion to a full vector of intrusions). Each scenario is tested 30 times, so the average number of false positives and false negatives is recorded, for each scenario, in order to find the Hanssen and Kuipers' score.

The average of the Hanssen and Kuipers' scores for each Equation is found and compared statistically using the Z score.

Data Analysis

First the Hanssen and Kuipers' scores for Equations 7.4 and 7.3 will be calculated for each scenario. After that, the corresponding average of the Hanssen and Kuipers' scores will be calculated for each Equation, and they will be compared looking at the Z score, in order to check if both are statistically significant different.

In order to corroborate the acurase of the Hanssen and Kuipers' average for Equations 7.4 and 7.3, the bootstrapping technique will be applied taken $1,000$ samples (samples of 25 Hanssen and Kuipers' means respectively).

In order to see if there is a statistically significant difference between the Hanssen and Kuipers' scores obtained with Equations 7.4 and 7.3, a combined bootstrapping will be applied using both sets (i.e., they comform a set of 50 combined Hanssen and Kuipers' scores) and the corresponding comparison is made with the average of the Hanssen and Kuipers' averages of each equation, in order to check if they are statitically diferent at the 95% level.

7.2.1 Results for Specific Hypothesis 2

30 times the GA with the dynamic fitness function as in Equation 7.3 was executed, with the set of GA parameters as the parameter set 2 and with the union operator, recording the number of false positives and false negatives. No false positives, nor false negatives were present, so the Hanssen and Kuipers' score was a perfect score of 1.

	Parameters		*Average*		
Equation	α	β	*False +*	*False -*	*H.&K. Score*
7.4	392.0	1.0	0.16	0.00	0.988
7.4	4.0	0.05	9.86	0.03	0.239
7.4	50.0	7.0	0.10	0.23	0.971
7.4	50.0	1.0	0.20	0.00	0.985
7.4	50.0	0.5	1.06	0.00	0.918

Table 7.10: Average of false positives, false negatives, and the corresponding Hanssen & Kuipers' Score given by Equation 7.4, using different setting for external parameters α and β. 30 runs per parameter setting.

30 times the GA with the fitness function as in Equation 7.4 was executed, with the set of GA parameters as the parameter set 2, recording the average number of false positives and false negatives and the Hanssen and Kuipers' score as is shown in Table 7.10.[5]

In order to compare Equations 7.4 and 7.3, a new test set over 30 runs was performed with the best external parameters available so far ($\alpha_1 = 392$, $\beta_1 = 1.0$ that correponds to the best Hanssen and Kuipers' score of 0.988), and various scenarios[6] as shown in Table 7.11. Equation 7.3 with no external tuning, and using the union operator, outperforms Equation 7.4 in all scenarios. However, as the best external parameters for Equation 7.4 were obtained with an intrusion vector of 11 intrusions, two new scenarios were tested:

1. 11 intrusions with no exclusive ones: Equation 7.3 gives no false positives nor false negatives, i.e., a perfect Hanssen and Kuipers score of 1. Equation 7.4 gives 3 false positives and no false negatives in the 30 runs, with a Hanssen and Kuipers score for the average of 0.992.

2. 11 intrusions with one exclusive one: Equation 7.3 gives no false positives nor false negatives, i.e., a perfect Hanssen and Kuipers score of 1. Equation 7.4 gives 30 false positives and 30 false negatives in the 30 runs, with a Hanssen and Kuipers score for the average of 0.832.

A complement set of tests as the one shown in Table 7.11 that covered from two intrusions to 23 intrusions, (i.e., two intrusions, three intrusions, and so forth until 23

[5]Other integer values for $1 < \beta < 7$ with $\alpha = 50.0$ were tested until the average of false negatives were 0.0 No one of those gives as result 0.0 for averages of false positive and false negatives. Less than 30 runs were done until the 0.0 average of false negatives appeared.

[6]Previous tests used an artificial vector of 11 intrusions.

	Equation 6.1 *with* $\alpha = 392.0$, $\beta = 1.0$						Equation 6.3 *with Union Operator*					
	Scenario						*Scenario*					
	No Intrusion		1 *Intrusion*		24 *Intrusion*		*No Intrusion*		1 *Intrusion*		24 *Intrusion*	
	False+	False-	False+	False-	False+	False-	False+	False-	False+	False-	False+	False-
Ave.	9.67	0.00	9.00	0.03	0.00	3.03	0.00	0.00	0.00	0.00	0.00	0.00
Std.	1.35	0.00	0.00	0.18	0.00	1.43	0.00	0.00	0.00	0.00	0.00	0.00

Table 7.11: Comparison between best fitness functions as shown in Table 7.10 and fitness function as in Equation 7.3. 30 runs per scenario per Equation.

	2	3	4	5	6	7	8	9	10	12	13
Ave.	7.16	1.93	2.00	2.06	0.23	2.06	2.00	2.00	1.93	0.26	0.01
Std.	0.37	0.36	0.00	0.36	0.77	0.18	0.00	0.00	0.36	0.45	0.30
	14	15	16	17	18	19	20	21	22	23	-
Ave.	0.23	0.30	0.26	0.33	0.20	0.30	0.23	0.30	0.40	0.00	-
Std.	0.43	0.46	0.45	0.47	0.40	0.46	0.43	0.46	0.49	0.00	-

Table 7.12: Average of false positive given by Equation 7.4. 30 runs per scenario.

intrusions—0, 1, 11, and 24 intrusions were already calculated as shown in Table 7.11) were performed using Equations 7.3 and 7.4, running each set of intrusions 30 times. No false positives, nor false negatives were given by Equation 7.3. Table 7.12 shows the average of the false positives given by Equation 7.4 which gives no false negatives for this test set, and Table 7.13 shows the corresponding Hanssen and Kuipers' scores.

Taking together all 25 scenarios, i.e., from no intrusion to 24 intrusions (no exclusive ones), the estimated average of the Hanssen and Kuipers' scores, for Equation 7.4, gives 0.840 with a standard deviation of 0.279 and the estimated standard error of the average of 0.056. As the average of the Hanssen and Kuipers' score of Equation 7.3 is 1.00 with standard deviation of 0.00 and estimated standard error of 0.00 then the difference between the two averages is $1.00 - 0.840 = 0.160$ with an estimated standard error of 0.056. The

Number of Intrusions												
0	1	2	3	4	5	6	7	8	9	10	11	12
-0.403	0.579	0.675	0.908	0.900	0.892	0.987	0.879	0.875	0.867	0.862	0.992	0.978
Number of Intrusions												
13	14	15	16	17	18	19	20	21	22	23	24	-
0.999	0.977	0.967	0.967	0.953	0.967	0.940	0.942	0.900	0.800	1.000	1.000	-

Table 7.13: Hanssen & Kuipers' scores given by Equation 7.4 for averages as in Tables 7.11 and 7.12.

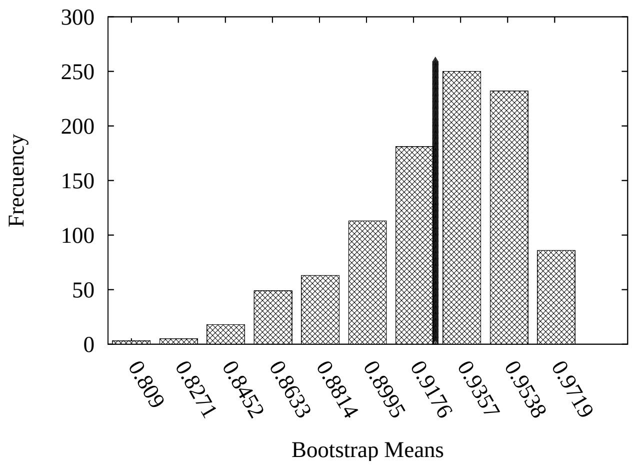


Figure 7.119: Distribution of the means of a randomized $1,000$ Hanssen and Kuipers' samples taken from 25 intrusion scenarios. Line corresponds to the mean 0.84 of the observed data. Original data from Equation 7.4.

observed difference 0.160 is $0.160/0.056 = 2.86$ estimated standard errors greater than zero, which means that the Hanssen and Kuipers' scores for Equation 7.4 is statistically significantly different than the average of the Hanssen and Kuipers' scores of Equation 7.3 at the 95% confidence level according to the Z score (Cohen, 1995).

In order to corroborate the accuracy of the estimated average of 0.84, bootstrapping was applied (Efron and Tibshirani, 1993): $1,000$ samples were taken at random from the 25 scenarios. The average of the $1,000$ sample averages was 0.8482 with an estimated standard error of 0.054, these values are quite similar to the ones previously calculated for the Hanssen and Kuipers' scores. Figure 7.119 shows the histogram of the mean of the $1,000$ samples means and the one that corresponds to 0.84.

Looking at the two Equations, if we combined the two data sets of Hanssen and Kuipers' scores obtained independently for each Equation, and perform the bootstrapping technique again over $1,000$ samples, an estimated averaged combined mean of 0.9258 with estimated standard error of 0.0284 is obtained. Comparing the combined estimated mean (0.9258) with the estimated mean of Equation 7.4 using the Z score, the result obtained is $(0.8482 - 0.9258)/(0.0559 - 0.0284) = -2.82$ which means that both means correspond to statitically significantly different distributions at the 95% confidence level. If the two estimated means that are compared are combined with the one that corresponds to the average of Equation 7.3 the Z score is $(1.00 - 0.9258)/0.0284 = 2.61$ which means

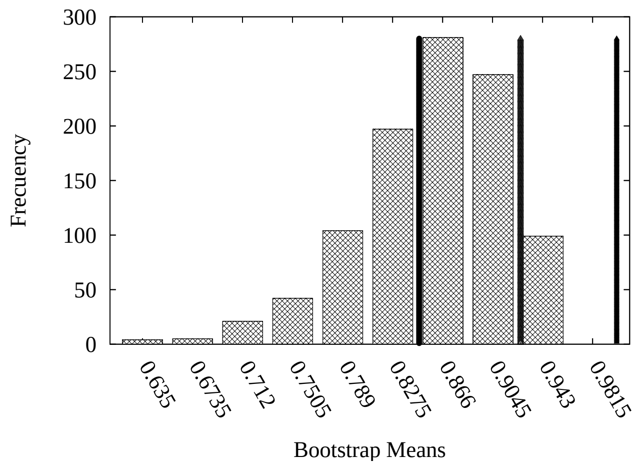


Figure 7.120: Distribution of the means of a randomized $1,000$ samples taken from 25 intrusion scenarios when original Hanssen and Kuipers' scores are combined for Equations 7.3 and 7.4. Left line corresponds to the mean 0.84 of the observed data for Equation 7.4. Middle line corresponds to the mean of the means over the $1,000$ samples of the combined data. Right line corresponds to the mean 1.00 of the observed data for Equation 7.3.

that the distributions of each estimated mean are statistically significantly different at the 95% confidence level. The corresponding histogram for the combined case is in Figure 7.120.

7.3 The Selection Pressure and Crossover and Mutation Probabilities Relationship

The third research question looks at the GA operators Selection pressure, crossover and mutation. The crossover operator has the purpose of recombining chromosomes in order to climb to a maximum according to the fitness function given. However, there are problems where the crossover operator could be quite destructive, for instance, the snake-in-the-box problem. The mutation operator gives the possibility of local changes; therefore, mutation helps with diversity and in moving the algorithm from stationary points. However, a high mutation rate could make the algorithm diverge and a low mutation rate could make the algorithm spend more generations to reach a maximum (see Section 6.3.6). But a GA with just crossover and mutation would be a random walk. Selection pressure is the invisible hand that makes GAs do progress in looking for possible solutions (Diaz-Gomez and Hougen, 2007f). However, a high selection pressure could make the algorithm to converge prematurely and a low selection pressure could make the algorithm to spend a lot of number of generation to find a possible solution (Jaroslaw Arabas and Mulawka, 1995). How then, should one set selection pressure (S_p), the probability of crossover (p_c) and the probability of mutation (p_m) so that GAs can be improved? Usually researchers and practitioners use empirical studies to approach this problem. We are proposing the use of the schema theorem in order to find ranges of possible values for the crossover and mutation probabilities depending on the problem to be solved, that could help the algorithm to obtain better solution quality. However, it should be highlighted that the fitness function and parameters of GAs are influencing the expected performance of the GA too.

The operators Ω: Is there a relation between selection pressure S_p and crossover and mutation operators of the corresponding plan τ, that influences the evolutionary process, and if there is, what is that relation?

General Form of Hypothesis 3.

There is a relationship between selection pressure (S_p), crossover (p_c) and mutation rates (p_m)

$$p_c \leq \frac{1}{\epsilon}\left(1 - \frac{1}{(1 - p_m h)S_p}\right) \tag{7.5}$$

with $p_m h \neq 1$, $p_m \approx 0$ but $\neq 0$, $h > 1$, $p_c \neq 0$ and $S_p > 1$, where ϵ is the crossover disruptive factorand h is the number of defining bits of the corresponding plan τ that influences the evolutionary process such that if $< S_p, p_c, p_m >$ satisfies Equation 7.5 and $< S'_p, p'_c, p'_m >$ does not satisfying Equation 7.5 then

$$X(G_A, < S_p, p_c, p_m >) \geq X(G_A, < S'_p, p'_c, p'_m >) \tag{7.6}$$

where $X(G_A, < S_p, p_c, p_m >)$ is the expected performance of a genetic algorithm G_A using parameters $< S_p, p_c, p_m >$. Expected performance is measured as the expected solution quality after a given number of generations.

$< S_p, p_c, p_m >$ ***parameter setting:***

S_p range is $2 \leq S_p \leq 6$ with step size 1, p_m range is $0.004 \leq p_m \leq 0.018$ with step size 0.002, and p_c range is $0.0 \leq p_c \leq 1.0$ with step size 0.1, in order to obtain the number of points indicated in Hypotheses $3.1.1$ and $3.2.1$.

Parameter justification: For the case of S_p, the values were chosen in that way because GAs usually work with $S_p > 1$, so we choose values from 2 with step size of 1. For the case of p_m the values were chosen ≥ 0.004 because the range $0 \leq p_m < 0.004$ are all in set $< S_p, p_c, p_m >$ that obey Equation 7.5 for the case of the one-max function, and we wanted to have a similar number of values in both sets (the ones that obey equation 7.5 and the ones that do not obey Equation 7.5) and for both problems (the one-max and the snake-in-the-box problems).

Parameters set 3.1 Population size $N = 100$, chromosome length $l = 100$, stop criterion when the GA reaches the global maximum or a maximum of 20 generations for the one-max problem.

Parameter Justification A similar set of parameters was suggested by Lobo and Goldberg (2004). They found that the algorithm usually converges to the global maximum in 25 generations. We reduced the number of generations to 20, in order to see the effect of S_p, p_c and p_m (avoiding ceiling effects). All other parameters are the same as those suggested by Lobo and Goldberg (2004).

Specific Hypothesis 3.1.1

Equation 7.6 is satisfied for the one-max problem using 297 triples $< S_p, p_c, p_m >$ satisfying Equation 7.5 and 143 triples $< S'_p, p'_c, p'_m >$ where Equation 7.5 does not hold. The set of parameters to use is parameter set 3.1.

Parameter set 3.2 population size $N = 1,000$, stop criterion when it reaches the global maximum in 100 generations for snakes in 8-dimensional hypercubes with a sub-snake of length 49 of a 7-dimensional hypercube embedded.

Parameter Justification A similar set of parameters was suggested by Bitterman (2004). We only reduced the number of generations to 100 in order to see the effect of S_p, p_c and p_m, avoiding ceiling effects.

Specific Hypothesis 3.2.1

Equation 7.6 is satisfied for the snake-in-the-box-problem using 167 triples $< S_p, p_c, p_m >$ satisfying Equation 7.5 and 273 triples $< S'_p, p'_c, p'_m >$ where Equation 7.5 does not hold. The set of parameters to use is parameter set 3.2.

Method

Run the algorithm 10 times for each set of triples $< S_p, p_c, p_m >$ (U) and $< S'_p, p'_c, p'_m >$ (V) recording the best solution found in the last generation. calculate the mean, the standard deviation, the maximun, the minimum and quartiles for each set over the 10 runs

Data Set	*Mean*	*Stdv.*	*Low*	Q_1	Q_2	Q_3	*High*
U	87.39	5.27	70	84	88	91	98
V	85.20	5.16	73	82	85	89	96

Table 7.14: Statistics for data set U satisfying Equation 7.5 and data set V that does not satisfy Equation 7.5. One-max problem. 10 runs per triple.

for each set U and V. Perform the Kolmogorov-Smirnov test. Perform the ANOVA test. Figures of the averages for each S_p will be drawn.

Data Analysis

Sets U and V will be compared using the Kolmogorov-Smirnov test in order to corroborate that the two sets U and V belong to statistically different distributions and, the ANOVA test will check the impact of each operator S_p, p_c and p_m independently, in pairs, and the three together, on the quality of the solution. Besides this, the mean, the standard deviation, the maximun, the minimum and quartiles for each set will be compared.

7.3.1 Results for Specific Hypothesis 3.1

297 triples of the form $< S_p, p_c, p_m >$ that satisfy Equation 7.5 (set U) and 143 triples of the same form that do not satisfy Equation 7.5 (set V) where generated. The algorithm was run 10 times for each set of $< S_p, p_c, p_m >$ generating 10 samples of 440 instances of quality of solution corresponding to each $< S_p, p_c, p_m >$. Table 7.14 shows the mean, standard deviation, highest, lowest, and quantities for both data sets. For the case of the data set that satisfies Equation 7.5, the three quartiles were higher for U than for V. The Kolmogorov-Smirnov test (K-S test) was applied over U and V. The corresponding p value was $9.07e-40$, the H value was 1, and the D value was 0.22 showing that the two data sets differs significantly (see Figure 7.121 where U and V are pointing the empirical cumulative distributions of U and V respectively, it can be observed that U is better than V).

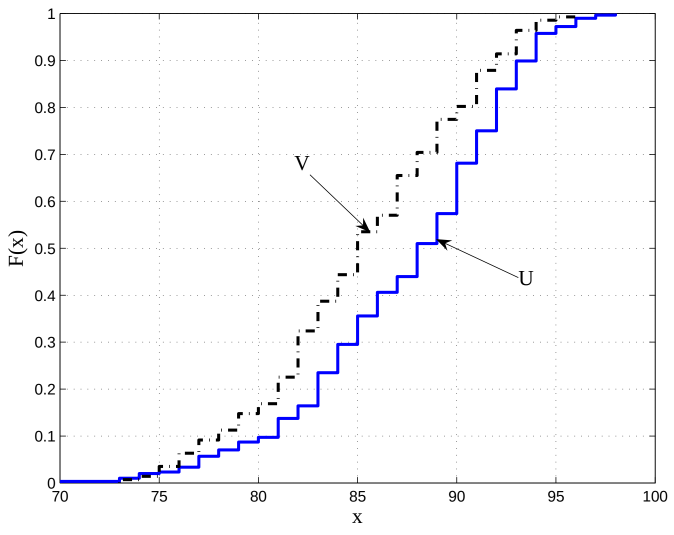


Figure 7.121: Empirical cumulative distributions of U and V at generation 20. One-max problem. x corresponds to fitness value.

For this problem we estimated $h = 50$ because initially the population is generated randomly, i.e., it is expected that there are approximately 50 ones and 50 zeros in each chromosome. If $p_c = 0$, then p_m has the higher value of $p_m = (S_p - 1)/S_p h$ (see Section 5.2.1), which for $S_p = 2$ gives $p_m = 0.01$ as initial upper value for the points that satisfy, and initial lower bound for the points that do not satisfy; and, for $S_p = 6$ gives $p_m = 0.017$ as upper bound for any point that satisfy. This means that p_m values greater than 0.017 do not satisfy Equation 7.5 (see section 6.3.6). However, the points that do not satisfy Equation 7.5, chosen to test these hypotheses where in the range of $0.01 \le p_m \le 0.018$ in order to perform a harder test.[7]

Figures from 7.122 to 7.126 show p_c and p_m together and the corresponding quality of the solution for different selection pressures according to parameter set 3.1. Figure 7.122 shows the case just discussed regarding the p_m probabilities selected for these tests, where p_m begins in 0.01 for the points that do not satisfy Equation 7.5. As the selection pressure is increased, the number of points that do not satisfy Equation 7.5, in the range chosen $0.01 \le p_m \le 0.018$ is lower because p_m, for the points that satisfy, is increasing according

[7]For example the triple $< 2, p_c, 0.01 >$ where $0.0 < p_c \le 1.0$.

to $p_m = (S_p - 1)/S_p h$ (see Figures from 7.122 to 7.126 where it is shown that is better for small schema, as this is the case, to have a higher p_c, a smaller p_m and a higher S_p).

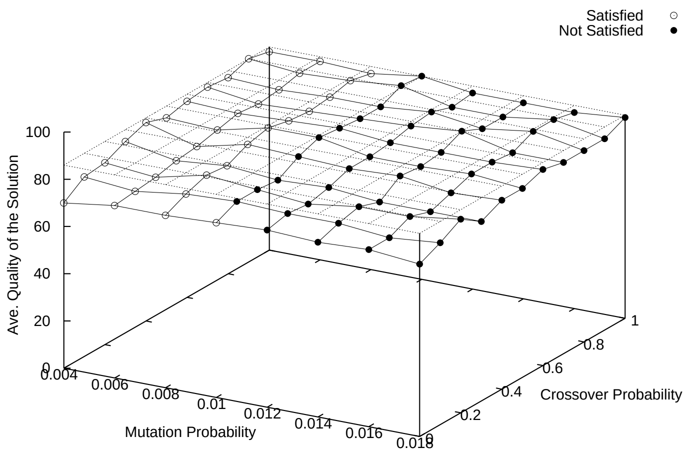


Figure 7.122: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 2$. One-max problem. Ave. over 10 runs at generation 20.

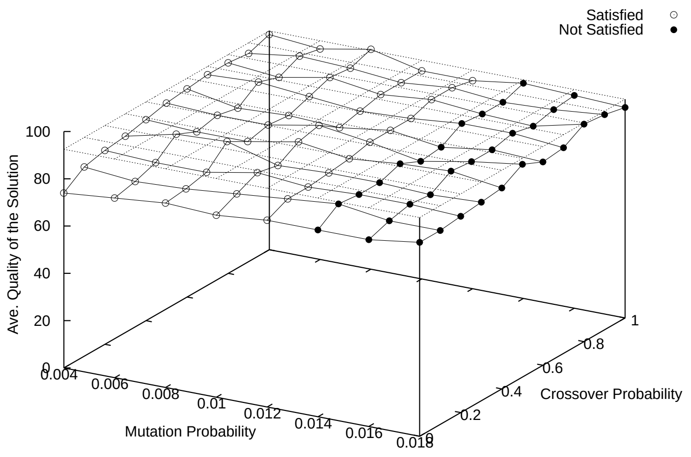


Figure 7.123: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 3$. One-max problem. Ave. over 10 runs at generation 20.

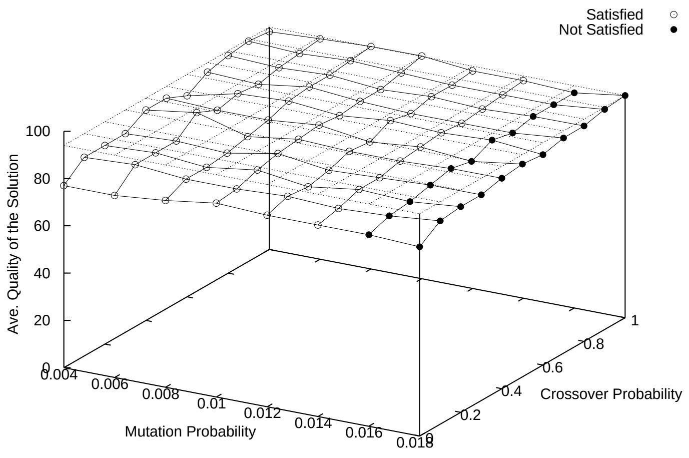


Figure 7.124: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 4$. One-max problem. Ave. over 10 runs at generation 20.

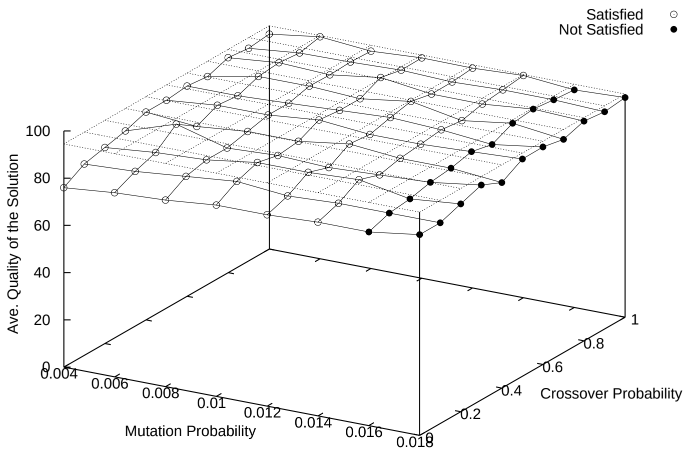


Figure 7.125: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 5$. One-max problem. Ave. over 10 runs at generation 20.

Source	*Sum. Sq.*	*d. f.*	*Mean Sq.*	*F*	$Prob > F$
p_m	1,115.4	7	159.3	9,225.38	0
p_c	57,486.3	10	5,748.6	332,815.17	0
S_p	54,477.6	4	13,619.4	788,491.73	0
$p_m * p_c$	2,605.1	70	37.2	2,154.63	0
$p_m * S_p$	1,674.1	28	59.8	3,461.53	0
$p_c * S_p$	1,186.5	40	29.7	1,717.35	0
$p_m * p_c * S_p$	6,327.8	280	22.6	1,308.38	0
$Error$	68.4	3,960	0	-	-
$Total$	124,941.3	4,399	-	-	-

Table 7.15: ANOVA test for the one-max problem at generation 20.

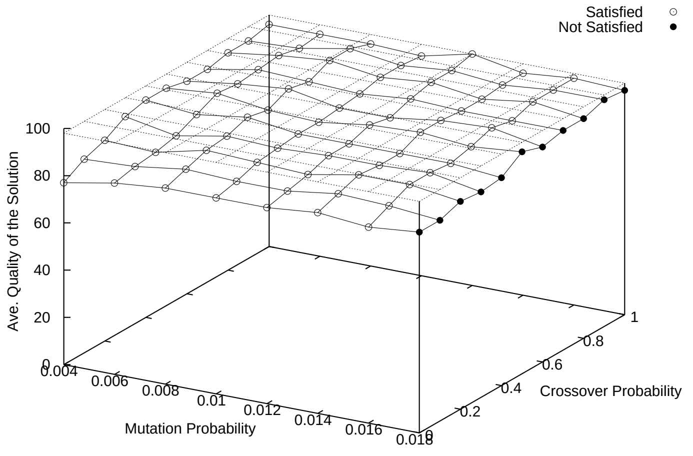


Figure 7.126: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 6$. One-max problem. Ave. over 10 runs at generation 20.

Table 7.15 shows the ANOVA results for this hypothesis. Each operator, p_c, p_m and S_p independently; each pair of operators, and the three operators together, have a statistically significant impact on the quality of the solution at much greater than 95% confidence level. S_p moderates the impact of p_m on the relationship between p_c and the quality of the solution.

Data Set	*Mean*	*Stdv.*	*Low*	Q_1	Q_2	Q_3	*High*
U	55.92	11.67	1	53	56	61	76
V	38.80	19.41	1	29.5	49	52	75

Table 7.16: Statistics for maximum length of snakes for data set U that satisfies Equation 7.5 and data set V that does not satisfy Equation 7.5. Snake-in-the-box problem in 8-dimensional hypercube. 10 runs per triple. Initial population randomly generated.

7.3.2 Results for Specific Hypothesis 3.2

167 triples of the form $< S_p, p_c, p_m >$ that satisfy Equation 7.5 and 273 triples of the same form that do not satisfy Equation 7.5 where generated. The algorithm was run 10 times for each set of $< S_p, p_c, p_m >$, generating 10 samples of 440 instances of quality of solution.

Three cases are going to be considered to test this hypothesis: (1) when the initial population is generated randomly, (2) when the initial population has half of the population with the schema the GA has to look for and half of the population is randomly generated, and (3) a third case where the entire initial population has the schema the GA has to look for. These three cases were considered in order to test the performance of the algorithm with different scenarios (initial populations) and sets U and V respectively.

Tables 7.16, 7.18 and 7.20 show the mean, standard deviation, highest, lowest and the corresponding quartiles for both data sets for each case: when the initial population is generated randomly, when half of the initial population has schema ξ and half is generated randomly, and when all the entire initial population has schema ξ. For data set U that satisfy Equation 7.5 each quartile was above the corresponding quartile when the algorithm uses $< S'_p, p'_c, p'_m >$ that does not satisfies Equation 7.5 (data set V) for all cases. The corresponding K-S tests were performed for each case: when the initial population is generated randomly $p < 5.50E-282$, $H = 1$, $D = 0.557$, when half of the initial population has schema ξ and half is generated randomly $p < 1.78E - 292$, $H = 1$, $D = 0.568$ and when all the initial population has schema ξ, $p < 4.47E - 294$, $H = 1$, and $D = 0.569$; showing that in all cases the two data sets U and V are statistically significantly different (see Figures 7.127, 7.128, and 7.129, where U and V are pointing to the empirical cumulative distributions of U and V, respectively; it can be observed that U is better than V).

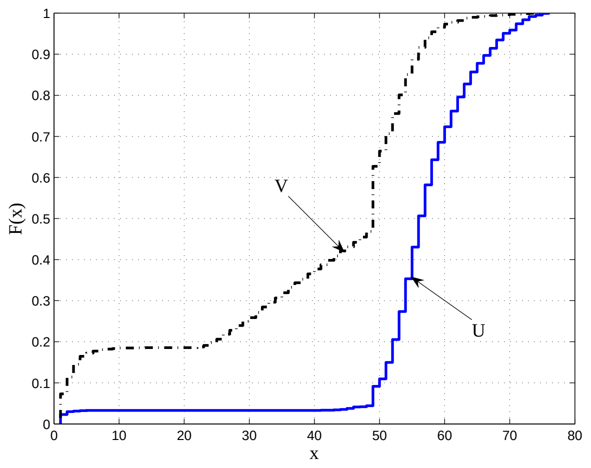


Figure 7.127: Empirical cumulative distributions of U and V at generation 100. Initial population randomly generated.

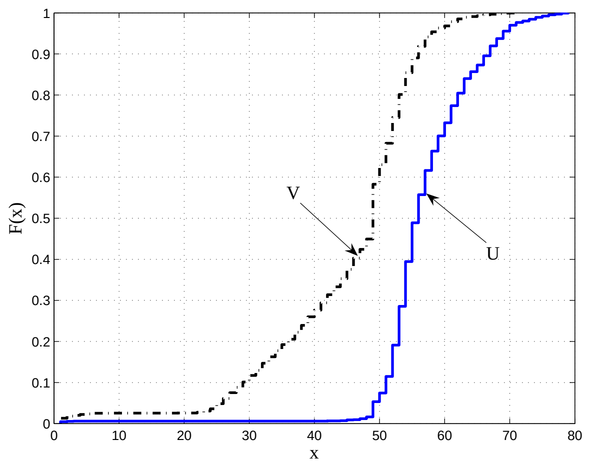


Figure 7.128: Empirical cumulative distributions of U and V at generation 100. Half of the initial population with schema included and half randomly generated.

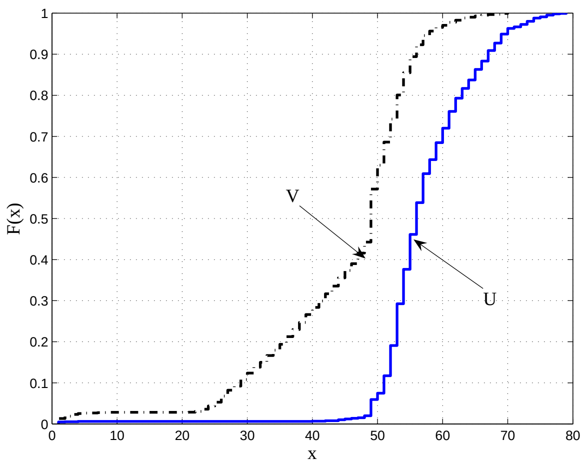


Figure 7.129: Empirical cumulative distributions of U and V at generation 100. Entire initial population with schema included.

Figures from 7.130 to 7.144 show p_c and p_m together and the corresponding quality of the solution for different selection pressures according to the set of parameters 3.2 for the three cases considered.

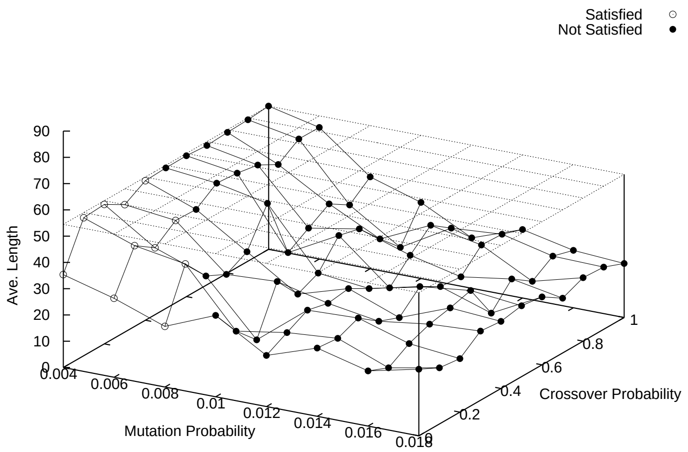


Figure 7.130: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 2$. Snake-in-the-box problem. Initial population randomly generated. Ave. over 10 runs at generation 100.

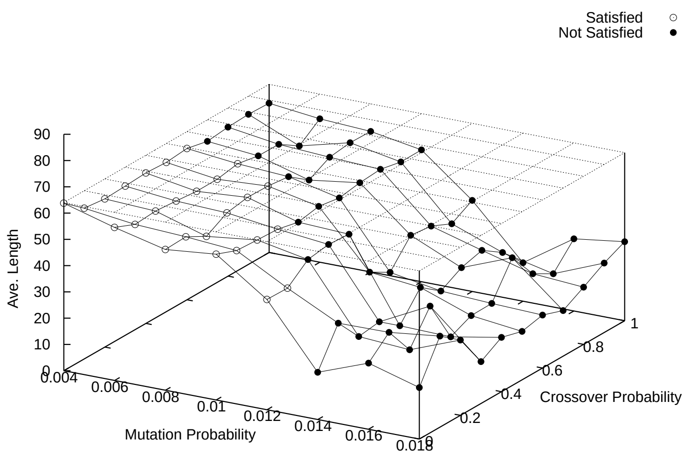


Figure 7.131: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 3$. Snake-in-the-box problem. Initial population randomly generated. Ave. over 10 runs at generation 100.

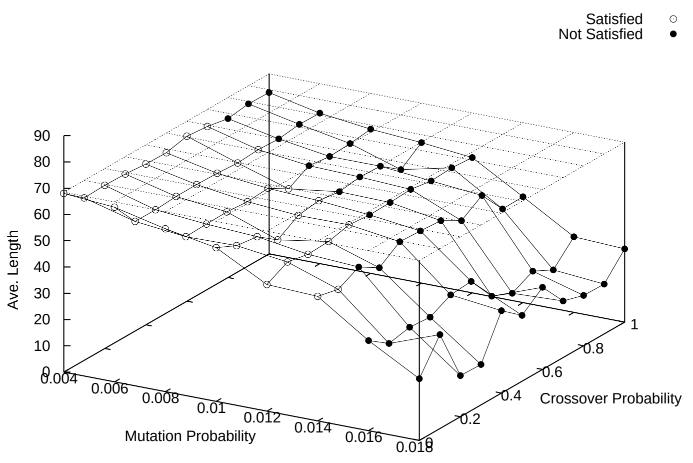


Figure 7.132: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 4$. Snake-in-the-box problem. Initial population randomly generated. Ave. over 10 runs at generation 100.

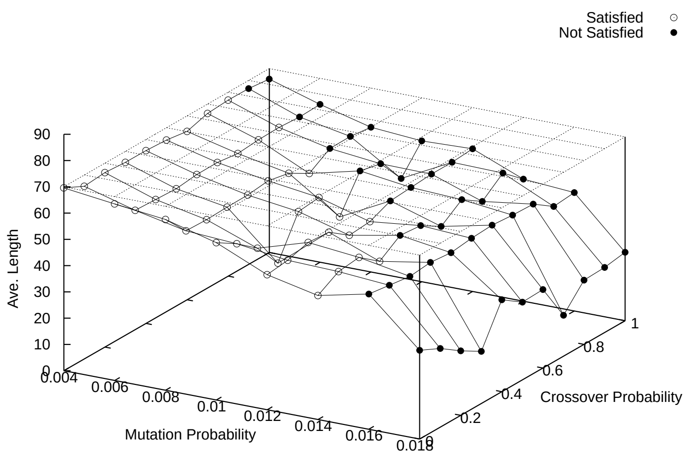


Figure 7.133: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 5$. Snake-in-the-box problem. Initial population randomly generated. Ave. over 10 runs at generation 100.

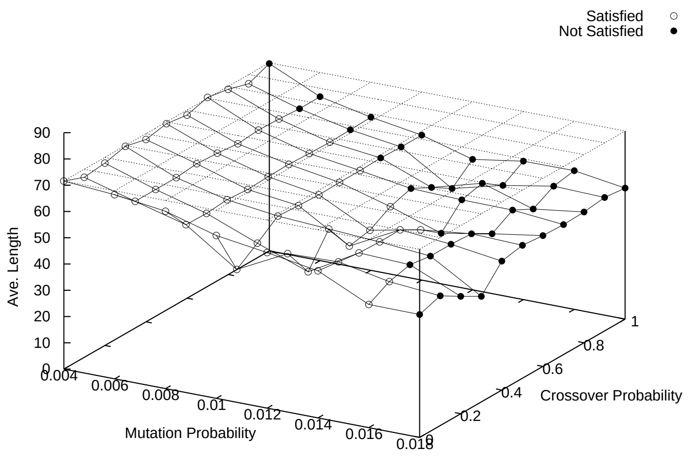


Figure 7.134: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 6$. Snake-in-the-box problem. Initial population randomly generated. Ave. over 10 runs at generation 100.

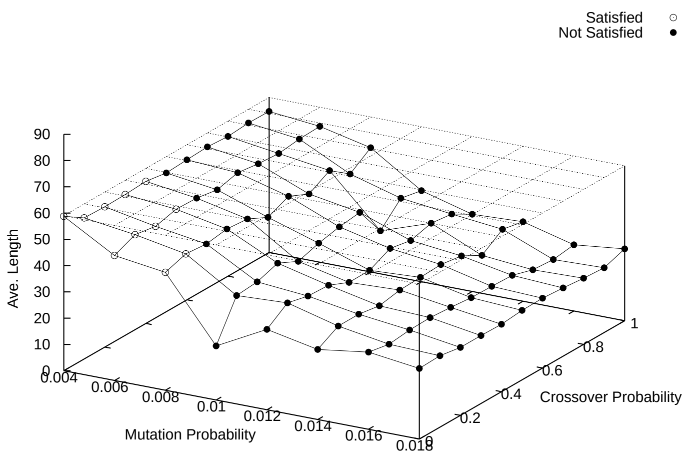


Figure 7.135: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 2$. Snake-in-the-box problem. Half of the initial population with schema included and half randomly generated. Ave. over 10 runs at generation 100.

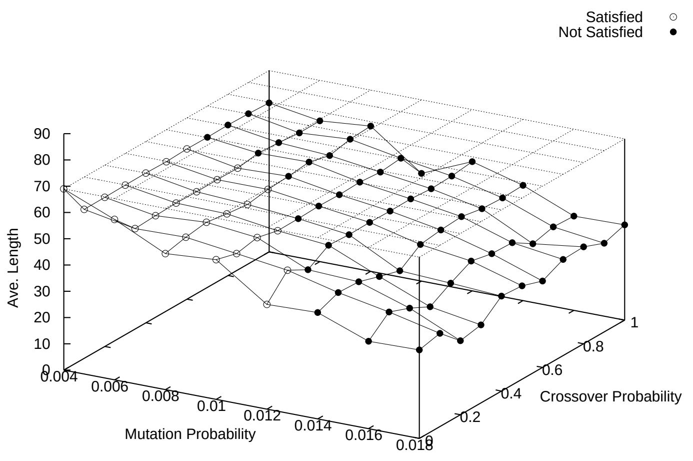


Figure 7.136: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 3$. Snake-in-the-box problem. Half of the initial population with schema included and half randomly generated. Ave. over 10 runs at generation 100.

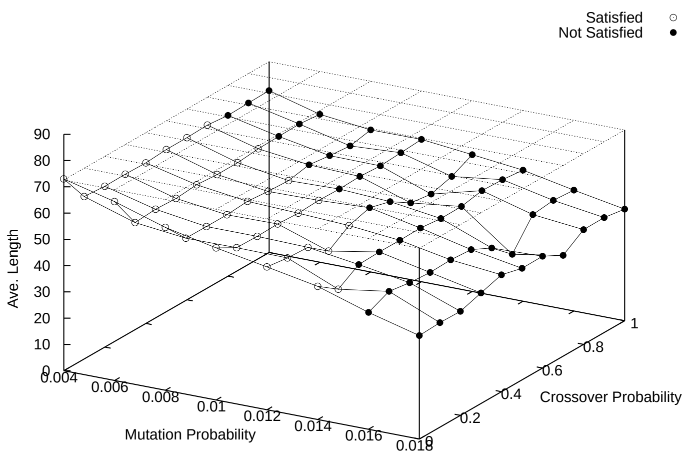


Figure 7.137: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 4$. Snake-in-the-box problem. Half of the initial population with schema included and half randomly generated. Ave. over 10 runs at generation 100.

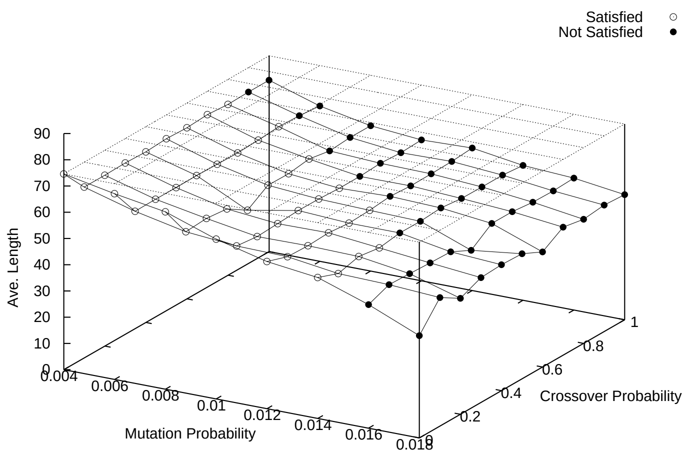


Figure 7.138: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 5$. Snake-in-the-box problem. Half of the initial population with schema included and half randomly generated. Ave. over 10 runs at generation 100.

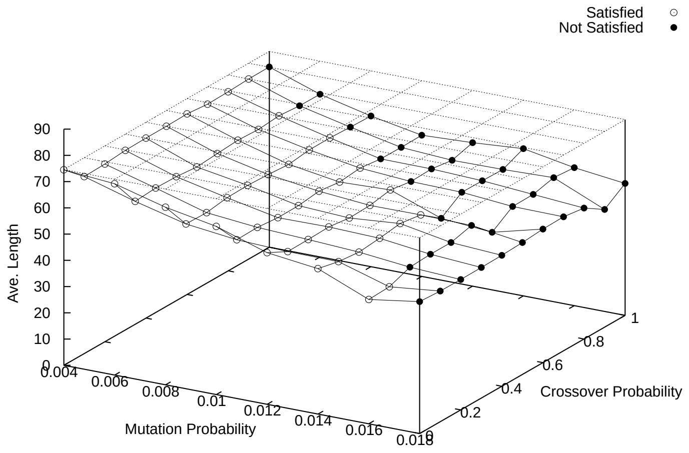


Figure 7.139: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 6$. Snake-in-the-box problem. Half of the initial population with schema included and half randomly generated. Ave. over 10 runs at generation 100.

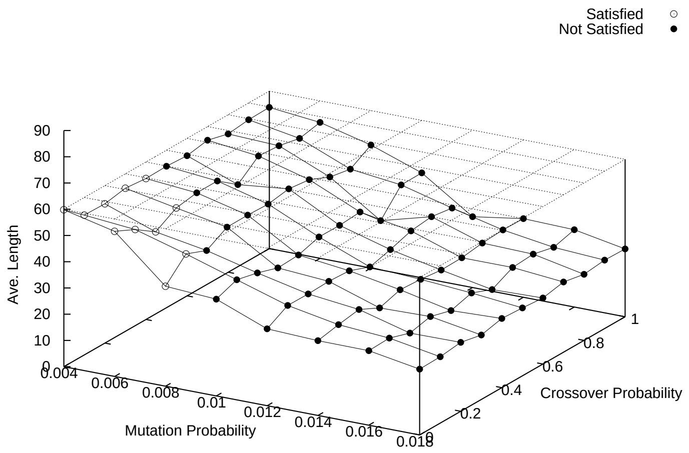


Figure 7.140: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 2$. Snake-in-the-box problem. Initial population with schema included. Ave. over 10 runs at generation 100.

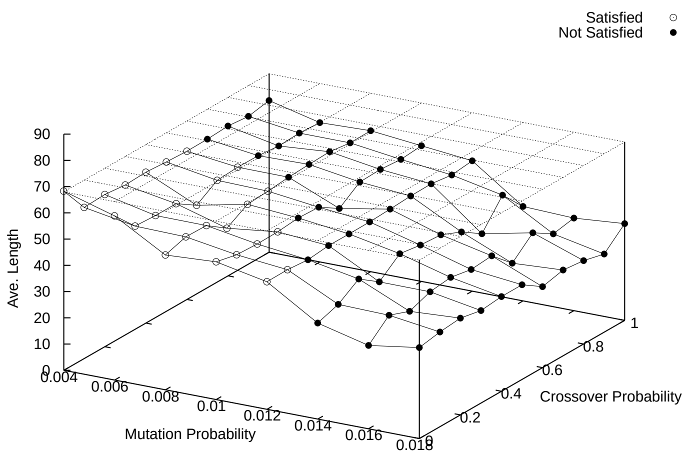


Figure 7.141: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 3$. Snake-in-the-box problem. Initial population with schema included. Ave. over 10 runs at generation 100.

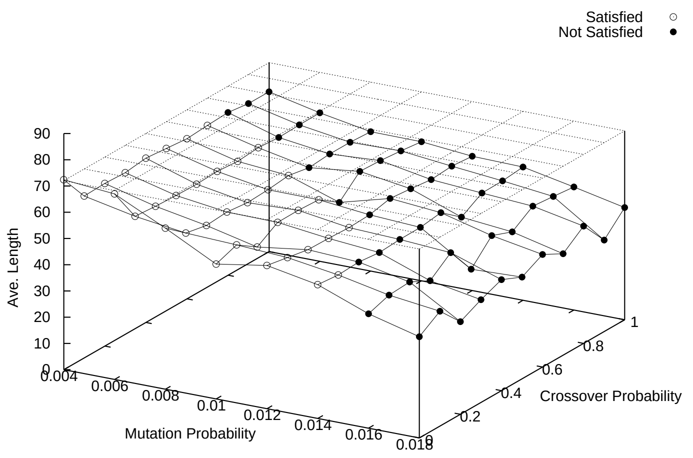


Figure 7.142: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 4$. Snake-in-the-box problem. Initial population with schema included. Ave. over 10 runs at generation 100.

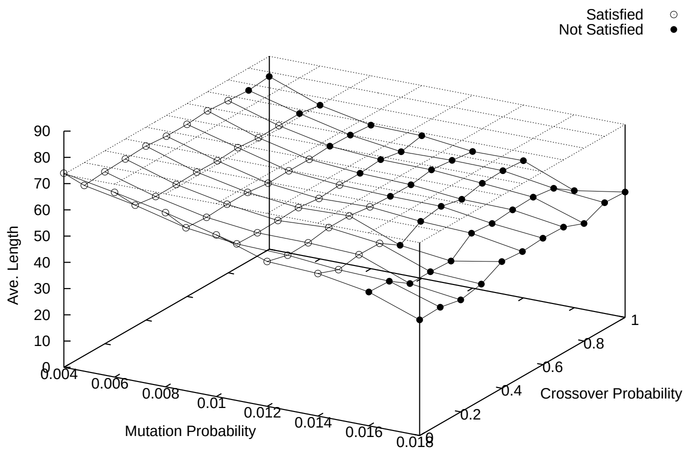


Figure 7.143: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 5$. Snake-in-the-box problem. Initial population with schema included. Ave. over 10 runs at generation 100.

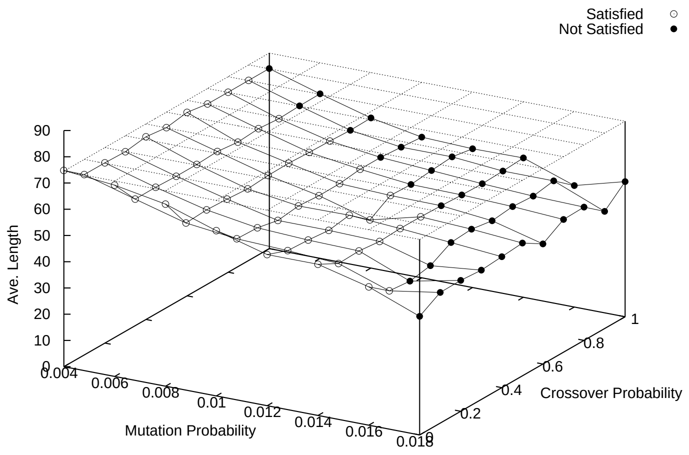


Figure 7.144: Probability of crossover and probability of mutation vs. quality of the solution. $S_p = 6$. Snake-in-the-box problem. Initial population with schema included. Ave. over 10 runs at generation 100.

Source	*Sum. Sq.*	*d. f.*	*Mean Sq.*	*F*	$Prob > F$
p_m	454,114.9	7	64,873.6	433.92	0
p_c	3,294	10	329.4	2.2	0.0151
S_p	327,984.6	4	81,996.1	548.45	0
$p_m * p_c$	15,165.7	70	216.7	1.45	0.009
$p_m * S_p$	112,379	28	4,013.5	26.85	0
$p_c * S_p$	10,007.4	40	250.2	1.67	0.0051
$p_m * p_c * S_p$	43,609	280	155.7	1.04	0.3101
$Error$	592,037	3,960	149.5	-	-
$Total$	1,558,591.6	4,399	-	-	-

Table 7.17: ANOVA test for the snake-in-the-box problem at generation 100. Initial population randomly generated.

Data Set	*Mean*	*Stdv.*	*Low*	Q_1	Q_2	Q_3	*High*
U	57.07	7.40	1	53	56	61	79
V	45.09	11.62	1	39	49	52	71

Table 7.18: Statistics for maximum length of snakes of data set U that satisfies Equation 7.5 and data set V that does not satisfies Equation 7.5. Snakes-in-the-box problem in 8-dimensional hypercube. 10 runs per triple each S_p. Half of the population with schema the GA has to look for and half of the population randomly generated.

Table 7.17 shows the analysis of variance for the first case. There is a statistically significant impact of each operator alone, and pair-wise in the quality of the solution, but not of the three operators together.

Table 7.19 shows the analysis of variance for the second case. There is a statistically significant impact of each operator alone, pair-wise, and all the three together in the quality of the solution.

Table 7.21 shows the analysis of variance for the third case, where the initial population has the schema ξ in all chromosomes. There is a statistically significant impact of each operator alone and pair-wise in the quality of the solution. However, there is no statistically significant impact looking at the three independent variables together.

Source	*Sum. Sq.*	*d. f.*	*Mean Sq.*	*F*	$Prob > F$
p_m	217,740.3	7	31,105.8	915.44	0
p_c	2167	10	216.7	6.38	0
S_p	188,113.7	4	47,028.4	1,384.04	0
$p_m * p_c$	7,274.7	70	103.9	3.06	0
$p_m * S_p$	21,234.9	28	758.4	22.32	0
$p_c * S_p$	4,552.3	40	113.8	3.35	0
$p_m * p_c * S_p$	13,325.5	280	47.6	1.4	0
$Error$	131,566.9	3,872	34	-	-
$Total$	583,200.1	4,311	-	-	-

Table 7.19: ANOVA test for the snake-in-the-box problem at generation 100. Initially half of the population with schema the GA has to look for and half of the population randomly generated.

Data Set	*Mean*	*Stdv.*	*Low*	Q_1	Q_2	Q_3	*High*
U	57.31	7.61	1	53	56	61	79
V	44.95	11.88	1	39	49	53	70

Table 7.20: Statistics for maximum length of snakes of data set U that satisfy Equation 7.5 and data set V that does not satisfy Equation 7.5. Snake-in-the-box problem in 8-dimensional hypercube. 10 runs per triple per each S_p. Entire initial population with schema ξ.

Source	*Sum. Sq.*	*d. f.*	*Mean Sq.*	*F*	$Prob > F$
p_m	238,216.4	7	34,030.9	869.8	0
p_c	3,981.6	10	398.2	10.18	0
S_p	198,001.4	4	49,500.4	1,265.19	0
$p_m * p_c$	7,870.6	70	112.4	2.87	0
$p_m * S_p$	21,737	28	776.3	19.84	0
$p_c * S_p$	2,807	40	70.2	1.79	0.0016
$p_m * p_c * S_p$	12,186.2	280	43.5	1.11	0.1035
$Error$	154,934.6	3,960	39.1	-	-
$Total$	639,735.2	4,399	-	-	-

Table 7.21: ANOVA test for the snake-in-the-box problem at generation 100. Entire initial population with schema the GA has to look for.

Chapter 8

Discussion

GAs, by definition, are algorithms; they receive an input, they process the input and they give a result. The result is the possible solution obtained and, as such, it would be beneficial to obtain better solutions. However, the quality of the solution for GAs depends on the parameters used: the number of generations or stop criterion, the type of crossover, the probabilities of crossover and mutation, the selection mechanism, and possibly the selection pressure. The problem, as addressed in Chapter 1, is then, how to choose good parameters that could have a positive impact in the quality of the possible solution. Regarding the input, a parameter to be set is the population size which has been related to diversity(Jaroslaw Arabas and Mulawka, 1995; Yu et al., 2006). A common belief is that, if the initial population is more diverse then it is expected that the quality of the possible solution is likely to be better (Jaroslaw Arabas and Mulawka, 1995; Bitterman, 2004; Burke et al., 2004; Lobo and Lima, 2005; Grefenstette, 1986; McPhee and Hopper, 1999; Rosca, 1995), so this dissertation tested this hypothesis.

The second hypothesis of this dissertation touches the core of GAs: fitness functions. Fitness functions guide GAs to possible solutions, they work in conjunction with the selection mechanism in choosing chromosomes that possibly are going to be mated and give rise to possibly better solutions. So, it is expected that if the fitness function is selected correctly, it could guide the algorithm to good solutions, however, if it is not set correctly, it can mislead the algorithm. The problem here is then, how to join objective(s) and constraint(s) in fitness functions that could help in the setting of fitness functions; the second hypothesis was tested in the context of this dissertation to address this problem.

But going a step forward this dissertation asked if there is a relationship between some parameters that could influence the performance of GAs. The third hypothesis tested a relationship between selection pressure S_p and the crossover (p_c) and mutation (p_m) probabilities using the schema theorem as its foundation.

This chapter analyzes the results obtained in Chapter 7 regarding the three hypotheses tested in this dissertation: the impact of diversity on the initial population in the performance of GAs, the impact in the use of internal parameters vs. external parameters that join the objectives and the constraints in fitness functions; and the impact of the relationship between the probability of crossover, the probability of mutation, and the selection pressure on the quality of the solution.

8.1 Hypothesis 1: Diversity in the Initial Population and Performance of GAs

One of the problems with diversity is how to measure diversity in a population. Diversity is the difference between the objects that are being compared. The first approach, then, is to look at the difference at the gene level. If the initial population is represented as a two dimensional matrix, gene level diversity can be measured looking at each column. A second approach is to look at diversity at each chromosome, in which case diversity can be measured looking at each row (see Chapter 5 for more details).

Once how to measure diversity is defined, this dissertation tested the first hypothesis that says in its general form:

$$\text{If } V(P_A) \geq V(P_B) \text{ then } X(G, P_A) \geq X(G, P_B), \tag{8.1}$$

where $V(P)$ is the diversity of population P and $X(G, P)$ is the expected performance of a genetic algorithm G with population P. Expected performance is measured as the expected solution quality of the best solution found so far after a given number of generations or the expected number of function evaluations to obtain a solution of a given quality.

Two problems, two diversity metrics, and two expected performance metrics were used, to give a total of eight specific hypotheses, which are summarized in Table 7.1 in Section 7.1. For none of the specific hypothesis tested was a strong correlation between diversity and performance found (see Section 7.1) using as test statistics the Pearson's coefficient and the Fisher's r to z transform.

The Pearson's coefficient R_{XY} gives a positive, negative, or no linear relationship between two variables (Cohen, 1995). For the particular case of Hypothesis 1, the two variables are diversity and performance of a GA (measured as the solution quality or number of generations to reach a global maximum). If $R_{XY} \approx 1.0$ then there is a strong to perfect positive linear correlation, and if $R_{XY} \approx -1.0$ there is a strong to perfect negative linear correlation (Cohen, 1995). No one of those values was reached for any of the specific hypothesis as can be seen in Tables 7.2, 7.4, 7.6, and 7.8 in Section 7.1. However, to be more certain about the independence of the two variables the Fisher's r to z transform was applied, the results of which are in Tables 7.3, 7.5, 7.7, and 7.9. In all cases the correlation was almost zero, and in one case, a small negative correlation was found (see Table 7.6 second trial at generation 110 the Pearson value of -0.227). Because no result returns a statistically significant outcome, we are not concerned with false positives of multiple comparison test, and the conclusion remains that there is no direct linear correlation between diversity and performance.[1]

Results for specific hypotheses 1.1 to 1.8 showed an independence between diversity in the initial population and performance of GAs. If diversity alone is not impacting performance of GAs, then what could be the factor or factors in the input that could be impacting performance? As was highlighted at the beginning of this chapter, population size is another parameter to be set when working with GAs. So, we return to the exploratory study where we found empirically a positive correlation between population size and performance (see Chapter 6). However, population size as addressed in Chapter 6 carried a change in diversity,[2] and this fact lead us to hypothesize that the factor that was impacting performance was diversity. What, then, is the implication of the negative result obtained for hypothesis 1? Possibly that diversity and population size together are factors that impact performance or that population size alone is a factor that is influencing performance. But, what about other factors? We need to consider not only the input, but the process that is in between the input and the output. As the algorithm iterates, usually the diversity measure is decreasing. If decreasing in diversity leads to a value near of zero quite soon, it is possible that a premature convergence occurs with poor quality (Jaroslaw Arabas and Mulawka, 1995; Burke et al., 2004; Lobo and Lima, 2005; Grefenstette, 1986). This is one

[1]The only case of a Pearson's value of -0.227 and Fisher's value of -2.15 does not change the conclusion in the sense that, this value could occur by chance, and contrary of what it was expected, a higher diversity value returned a lower outcome.

[2]Usually increasing population size carries an increase in diversity in terms defined in Section 5.0.2.

of the difficulties when working with GAs—due to the interdependence of parameters, a change in one can impact the whole result (Eiben et al., 1999).

8.2 Hypothesis 2: Joining Objective(s) and Constraint(s) in Fitness Functions

A genetic algorithm needs a fitness function that combines the objective(s) and constraint(s) into a single value (Coello, 1998; Mé, 1998). The problem is not only to find the appropriate function but also to provide accurate values to the parameters that produce the correct solution of the problem for as many instances as possible. It appears that the fitness function proposed in *GASSATA*, a combination of objectives and constraints into a single value using arithmetical operations, should be correct. However, it is difficult to set the parameters so that the algorithm finds intrusions and converges (see Sections 3.3.1 and 5.1).

For this problem *GASSATA* uses a fitness function that has three terms: $\sum_{i=1}^{N_a} W_i I_i$ that is rewarding, $\beta\, T^2$ that is penalizing, and α that is used to balance the previous terms trying to maintain a positive value.[3]

As the fitness function is giving a payoff to the highest valued individual, the term $\sum_{i=1}^{N_a} W_i I_i$ is guiding the solution to have the maximum number of intrusions. However, this is good enough until the correct set of intrusions are found. Later on, i.e., if more intrusions than that are hypothesized, the problem of false positives occurs. Now, the penalty term $\beta\, T^2$ is diminishing the fitness value, but in doing so, various intrusions can require the same event, a fact that should be taken into account to avoid a higher number of false positives (Diaz-Gomez and Hougen, 2005c). These two facts were tested and false positives and false negatives were found (Diaz-Gomez and Hougen, 2005a).

In order to illustrate one more time some of the problems in doing external tuning to parameters that join objectives and constraints in fitness functions, see Table 7.10 in Section 7.2.1 that shows some examples of external parameters used in Equation 7.4. If the set of parameters (α, β) is $(50.0, 7.0)$ then there are not so many false positives but the average number of false negatives is the maximum (0.23). This is because as $\beta = 7.0$ is high (compared with others values used in these tests as the penalty factor) then the

[3]It is desirable to have positive values in fitness functions overall if proportional selection is used.

algorithm tries to avoid to hypothesized intrusions, because if it does it and it is wrong, then it is going to be highly penalized. Now, if $\beta = 7.0$ is high as a penalty factor we can try with lower values. Let say that $\beta = 1.0$, what happens then? The average of false negatives decreases to 0.0, but the average of false positive increases from 0.10 to 0.20. This is one of the problems of external tuning for fitness functions, the weighting of objective(s) and penalty term(s). On the other hand, α and β must be tuned for each input.

To solve the problem of external tuning for the intrusion detection problem, hypothesis 2 proposes three things:

- replace the positive side $\sum_{i=1}^{N_a} I_i$ that is guiding the algorithm to have false positives, with the union operator,
- penalize each chromosome taken into account if more than one intrusion requires the same event, and
- cut the α term.

The fitness function proposed by hypothesis 2 works in conjunction with the union operator to find the maximum number of intrusions and has only the penalty term:

$$F(I) = 1 - \left(\frac{1}{\sum_{j=1}^{N_e} (AE * I)_j} \right) \sum_{j=1}^{N_e} max[0, (AE * I)_j - OV_j] \tag{8.2}$$

where AE is the attack-event matrix, I is the chromosome, OV is the observed vector, $\beta = 1/\sum_{j=1}^{N_e}(AE * I)_j$, and $Penalty = \sum_{j=1}^{N_e} max[0, (AE * I)_j - OV_j]$.

Equation 8.2 does not need external tuning. The only parameter to set is β, but β is an *adaptive* parameter that depends on I. The drawback of Equation 8.2 is that it does not have have a rewarding term that pushes $F(I)$ to get more intrusions. For solving this drawback, the union operator was proposed (see Section 3.3.2).

The results obtained for hypothesis 2 showed a perfect Hanssen and Kuipers' score ($= 1$) with Equation 8.2 and the union operator, for all the 25 scenarios, outperforming Equation 7.4 which obtained on average a Hanssen and Kuipers' score of 0.840. In order to check if there is any difference between the Hanssen and Kuipers' score for each Equation and a combination of both scores, a bootstrapping approach was applied with the scores combined. There was statistically significant difference, at the 95% confidence level, between the combined scores and each one (see Section 7.2 where is shown specific results). This means, that the use of internal parameters that join the objective(s) and the constraint(s), in fitness functions, could be benefitial for finding better solutions or at least the external tuning, which sometimes can mislead the GA for finding a correct solution (Diaz-Gomez and Hougen, 2005c), is avoided.

8.3 Hypothesis 3: Interrelation Between $\mathbf{p_c}$, $\mathbf{p_m}$ and $\mathbf{S_p}$

It has been a common practice to use in almost all binary GAs the same set of parameters for crossover and mutation ($p_c = 0.6$ or $p_c = 0.7$ and mutation per bit $p_m = 0.001$), or, in some cases a trial and error procedure is used—researches try with different probabilities and the ones that give better results are the ones that are reported (Eiben et al., 1999).

Hypothesis 3 pointed out this problem and proposed a relationship between p_c, p_m and S_p as:

$$p_c \leq \frac{1}{\epsilon}\left(1 - \frac{1}{(1 - p_m h)S_p}\right), \tag{8.3}$$

with $p_m h \neq 1$, $p_m \approx 0$ but $\neq 0$, $h > 1$, $p_c \neq 0$ and $S_p > 1$, where ϵ is the crossover disruptive factor, h is the number of defining bits, and S_p is the selection pressure. Clearly, if $p_m = 0$ and $p_c = 0$ all what we have is selection pressure $S_p > 1$, so, if the solution is not in the initial population, then the GA is unable to find it (see Section 6). Cases for $p_m \approx 0$ and $p_c \approx 0$ remains an open question for future work.

Equation 8.3 shows that the choosing of p_c, p_m and S_p depends on the problem itself (ϵ and h) and that there is an interrelation between $< p_c, p_m, S_p >$ that could affect the quality of the solution.

Equation 8.3 was derived from the Schema Theorem (see Section 2.3.1), where it is expected that the number of schema ξ at step t is not going to decrease at step $t + 1$. But what is schema ξ? Schema ξ is the schema the algorithm is looking for. But, as it is almost certain that at the beginning schema ξ is not present, then the GA has to build it progressively at each time step. However, in the building process, destructive factors occur owing to crossover and mutation (see Sections 5.2.1 and 6.3.6), so the probabilities of crossover and mutation should be such that they try to be set avoiding that destructive effect, and this is what hypothesis 3 is proposing with Equation 8.3. However, an optimal setting of parameters for GAs is so difficult that trying to find them is almost impossible (Eiben et al., 1999).

Two problems were chosen to test the third hypothesis: the one-max problem and the snake-in-the-box problem. The one-max problem is such that the schema under consideration is small ($\epsilon = 1/(l-1)$) and the snake-in-the-box-problem in a 8-dimensional hypercube is such that $\epsilon = 110/127$ (that could be considered long) because a 50-long snake that begins at node 0 and finishes at node 110 was embedded in the lower hypercube. The algorithm needs then, to preserve the sub-snake given and look at the 128 nodes of the upper hypercube to complete it.

The set of triples $< p_c, p_m, S_p >$ chosen to perform the tests of specific hypotheses were such that there is almost no boundary between the points that satisfy Equation 8.3 (set U) and the points that do not satisfy Equation 8.3 (set V), making the problem harder to solve. The probability of crossover could be set in the full range $0.0 \leq p_c \leq 1.0$. The probability of mutation began at 0.004 because for the case of the one-max problem all the points with $p_m \leq 0.004$ belong to set U (330 in total independent of p_c and S_p), and we wish to have sets of approximately equal size. For example, for $S_p = 2$, the points that belong to V begin at $p_m = 0.01$, so the range for p_m was between 0.004 and 0.018.

For the case of the one-max problem, as the schema is small, it is expected that the crossover operator is not destructive. This fact is so strong that the algorithm perform quite well with p_c high in both sets U and V (see Figures 7.122 to 7.126 in Section 7.3.1). However, on average set U that obeys Equation 8.3 outperforms set V that does not obey Equation 8.3 as is shown in Table 7.14 in Section 7.3.1.

In order to check the impact of each operator alone, the impact of each pair-wise combination, and the impact of the three operators together on the quality of the solution, the ANOVA test was performed. The ANOVA test shows the effect of the independent variables over the dependent variable, as well as if there are some interactions of the independent variables that affect the dependent variable. Following Jaccard (1998), in order to perform the analysis of variance, it is useful to define a focal independent variable, a first order moderator independent variable and a second order moderator independent variable. It is a common belief that the crossover operator is the principal factor of influence in GAs and that the mutation operator is a background operator (Holland, 1992). So, we propose the crossover operator as the focal independent variable, the mutation operator as the first order moderator, and the selection pressure as the second order moderator. This helps us to analyze the three way interaction that could exist between crossover, mutation, and selection pressure.

Table 7.15 in Section 7.3.1 shows the ANOVA results for the one-max problem. Each operator, p_c, p_m and S_p alone; each pair-wise set of operators; and the three operators together impact the quality of the solution. S_p moderates the impact of p_m on the relationship between p_c and the quality of the solution.

For the case of the snake-in-the-box problem in a 8-dimensional hypercube, three cases were considered: (1) when the initial population is generated randomly[4] (possibly there is no presence of schema ξ in the initial generation), (2) when the initial population has half of the individuals with schema ξ and half are randomly generated, and (3) when all the individuals in the initial population have schema ξ. These three cases were considered in order to see the performance of the GA not only when schema ξ is present, but also, when the algorithm has no instances of it.

For the first case, where there is no presence of schema ξ, S_p plays its roll where it is more beneficial for the GA to have higher selection pressures for both sets U and V (see Figure 7.130 for selection pressure of 2 and Figure 7.134 for selection pressure of 6). Theoretically, p_c should be quite destructive but initially schema ξ is not present, so p_c is playing its roll of building ξ. However, it should be taken into account that for $3 \leq S_p \leq 6$ on average the maximum quality of solution was obtained when $pc = 0.0$ and $p_m = 0.004$ (a point belonging to set U). Table 7.16 shows the statistics for set U and V. On average, set U outperformed set V.

Table 7.17 shows the analysis of variance for specific hypothesis $3.2.1$ first case. There is a statistically significant impact of each operator alonea nd in pair-wise combinations on the quality of the solution, but not the three operators together. The ANOVA test is showing, then, that the interaction effect between S_p, p_c and p_m on the dependent variable is not statistically significant. It is possible that the cause is that the initial schema ξ that the algorithm is looking for is not initially present so it is expected that the majority of individuals are going to have approximately the same fitness value. The effect that could have on S_p in the moderation of the impact of p_m on the relationship between p_c and the quality of the solution is not statistically significant.

[4]It should be highlighted that a 50 sub-snake is embedded, and that just the last 128 genes are generated randomly.

For the second case, half of the initial population has schema ξ and half are generated randomly. In this case, the crossover operator could be destructive if an instance of schema ξ is selected to mate with an individual randomly generated. However, this fact is lesser in the sense that individuals with schema ξ probabilistically tend to be the ones selected (individuals with schema ξ are expected to be more fit). This time again, on average the maximum average of quality of solution was obtained when $p_c = 0$ and $p_m = 0.004$, and set U outperformed set V as is shown in Table 7.18.

Table 7.19 shows the analysis of variance for hypothesis $3.2.1$ second case. There is a statistically significant impact of each operator alone, in pair-wise combinations, and all the three together on the quality of the solution. The ANOVA test is showing then that there is a statistically significant correlation between the operators p_c, p_m and S_p that impact the quality of the solution, and it is possible that the cause is that the schema ξ the algorithm is looking for is initially present. In this case, schema ξ competes with different schema and the three operators p_c, p_m and S_p interact together to impact the quality of the solution.

For the third case, where all individuals in the initial population have schema ξ, the disruptive factor of p_c on ξ is not present. This does not mean that p_c has no impact on the solution quality because the algorithm nonetheless has to look for a snake in the 8-dimensional hypercube that contains the sub-snake given. This time again, on average the maximum average of quality of solution was obtained when $p_c = 0$ and $p_m = 0.004$, and set U outperformed set V as is shown in Table 7.20.

For cases where schema ξ is already present, the mutation operator could be quite destructive, such as a change of a bit in the $h = 50$ positions already fixed because of the snake embedded causes schema ξ to be destroyed.

Table 7.21 shows the analysis of variance for hypothesis $3.2.1$, third case, where the initial population has the schema ξ in all chromosomes. There is a statistically significant impact of each operator alone, and in a pair-wise combinations, on the quality of the solution. However, there is no statistically significant impact looking at the three independent variables together. It is expected that if all the individuals already have schema ξ, then almost all of them are going to be evaluated approximately with the same values, making it harder for S_p to moderate the impact of p_m in the relationship between p_c and the solution quality.

Chapter 9

Conclusions

Some questions arise when solving a problem with GAs: What is a good fitness function, a good number of individuals in the initial population, a good selection mechanism and selection pressure, a correct crossover probability, a correct mutation probability, and an appropriate stop criterion to use in order to solve a problem? The setting of these parameters constitutes an optimization problem (Diaz-Gomez and Hougen, 2007c; Eiben et al., 1999). Any set of values for these parameters: fitness function,[1] number of individuals, selection mechanism, selection pressure S_p, crossover probability p_c, mutation probability p_m, and stop criterion used, can lead the algorithm to a different quality of the solution (Grefenstette, 1986). Therefore, it is expected that the time spent in the selection of these parameters could be compensated for the performance of GAs, where performance is measured in solution quality and possibly in the number of generations to obtain a solution (Eiben et al., 1999).

This dissertation has worked three research questions that gathers five parameters: the diversity of the initial population, the fitness function, the crossover and mutation probabilities and the selection pressure, in an effort to address the difficult problem of optimization of parameters for GAs. The GA's parameters constitute, in some cases, open problems like the number of individuals in the initial population (Piszcz and Soule, 2006a) and all are problem dependent. However, our study approaches the initial population problem, proposing some measures of diversity in an effort to validate the general hypothesis that says that if the diversity measure $V(P_A)$ for a population P_A is greater than the diversity

[1]The fitness function defines the optimization problem, but in a practical sense it is a necessary step in the implementation of GAs, for example, in the snake-in-the-box problem, multiple fitness functions try to find the longest snake, but each one influences the finding of a better solution.

measure $V(P_B)$ for a population P_B of the same number of individuals, then, it is expected that the performance ($X(G, P_A)$) of the GA with population P_A is going to be better than the performance ($X(G, P_B)$) of the GA with population P_B. The problem of finding a good fitness function has been addressed by proposing the use of fitness function's internal parameters that penalize, in a graduated way, violations of the constraint, diminishing the false positive ratio. For the case of the crossover and mutation probabilities, some upper thresholds that could be used depending on the disruptive factors of these operators and selection pressure used, has been proposed, and a relationship between them has been established.

9.1 Diversity in Initial Population and GA Performance

The first general hypothesis points out the relationship between diversity in the initial population and the quality of the possible solution. This dissertation reports how to measure diversity in an initial population. Two approaches were used in order to measure diversity for the specific hypotheses 1.1 to 1.8: entropy and Hamming diversity, i.e., diversity at the gene-level and diversity at the chromosome-level. The results of these specific hypotheses showed no direct correlation between diversity and the quality of the solution, nor for the number of generations to reach the global maximum, for the one-max function and the snake-in-the-box problem in a 5-dimensional hypercube, using the set of parameters 1 (see Section 7.1). This result goes against the common belief that diversity in the initial population (using wide used definitions of diversity) influences directly the performance of GAs (Jaroslaw Arabas and Mulawka, 1995; Burke et al., 2004; Lobo and Lima, 2005; Grefenstette, 1986; McPhee and Hopper, 1999; Rosca, 1995). This does not mean that diversity in the initial population is not important, as stated in the exploratory study (see Chapter 6). But different than the exploratory study in Section 6.1.2, in hypothesis 1, population size is constant and diversity is the independent variable that could impact the performance GAs.

The rejection of hypothesis 1 lead us to think that maybe, it is not diversity alone but the increasing of the population size that could help sometimes in the performance of GAs as was shown in the empirical studies in Section 6.1.2 and Diaz-Gomez and Hougen (2007a). However, population size and diversity cannot be considered isolated from the rest of parameters, and specifically it is quite important to measure how diversity changes as the algorithm iterates. If diversity decreases quite soon then it is expected that the algorithm is going to converge prematurely. The parameter that could have a strong relation with diversity is selection pressure (Jaroslaw Arabas and Mulawka, 1995). If selection pressure is high then it is expected that the GA is going to choose the more fit individuals, and in consequence it is going to converge prematurely (Jaroslaw Arabas and Mulawka, 1995). If, on the contrary, selection pressure is quite low, then it is possible that the algorithm is going to expend a lot of computations to converge or it may diverge because of the lack of selection pressure (Jaroslaw Arabas and Mulawka, 1995). It should be highlighted that the selection pressure used for testing hypotheses 1.1 to 1.8 was previously used by Lobo and Goldberg (2004) ($S_p = 2$) in testing the one-max function and by Diaz-Gomez and Hougen (2006b,d) in testing the snake-in-the-box problem in 4 and 8-dimensional hypercubes. In conclusion, diversity cannot be seen alone in the initial population, but in conjunction with the rest of parameters: selection pressure (as described previously), crossover and mutation, that are responsible for exploring new regions of the search space, and the stop criteria, where diversity itself can be used as a measure of stopping.

9.2 Internal vs. External Parameters

The second general hypothesis addressed the difficult problem of finding good parameters that join the objective(s) and the constraint(s). For doing that, the corresponding empirical study for the intrusion detection problem and the snake-in-the-box problem was performed (see Section 6.2). Equation 7.3 penalizes each individual in a finer way with no use of external parameters. This characteristic makes Equation 7.3 good for avoiding false positives. Equation 7.3 with the union operator, free of external tuning, outperforms Equation 7.4, with external tuning, using the Hanssen and Kuipers' score (see Section 7.2) validating specific hypothesis 2.1 that says that Equation 7.2 is satisfied for the off-line intrusion detection problem, using as fitness function F_i Equation 7.3 with internal parameters and the union operator; and using as fitness function F_e Equation 7.4, and parameter set 2 (see Section 7.2). As Equation 7.3 alone has the drawback of giving false negatives, the union operator was suggested, a mechanism that stores the current solution and begins to add possible solutions if the algorithm finds them—because sometimes the algorithm could converge to a local maximum (see Section 6.2). However, if parameter set 2 (as suggested by Mé (1993)) is changed and the length l of the individual I is enlarged to hundreds or thousands, as done in the exploratory study (see Section 6.2.2.1), the false negative problem appears again. An increase in the number of individuals in the initial population was needed in order to diminish the false negative ratio (Diaz-Gomez and Hougen, 2007e).

Certainly, Equation 7.3, which is free of external tuning of parameters and outperforms Equation 7.4 (with external tuning), is a contribution. Work done in this dissertation tried to generalize the concept of Equation 7.3 using it partially for finding snakes in hypercubes (see Sections 5.1.3 and 6.2.3). However, the snake-in-the-box problem has certain characteristics, like the fact that the solution should be a connected path, that Equation 7.3 alone can not handle. Therefore, a new term in the fitness function was used: $Length(S)$. It should be highlighted that for the case of the snake-in-the-box problem there is no explicit observed vector OV that can be used in the penalty term in order to compare its entries with $(AM * S)_j$; therefore, because a node of the snake is such that $1 \leq (AM * S)_j \leq 2$, then the comparison is made as $\sum_{j=1}^{2^d} max[0, (AM * S)_j - 2]$, where AM is the adjacency matrix that encodes the d-dimensional hypercube and S is the chromosome (possible snake), or a count of the number of failures is taken, i.e, the number of times that $(AM * S)_j > 2$ for all $0 \leq j \leq 2^d$ such that $S_j = 1$. Besides the counting of bad points, the penalty term includes the counting of isolated and lazy points. Equation 6.5, normalized as Equation 7.3, is good for finding local minima, it is quite effective for finding snakes, but not longer ones (see Section 6.2.3). In an effort to obtain longer snakes, an empirical study regarding the mutation rate for snakes in 4-dimensional hypercubes was conducted, giving as a result a mutation rate as in Equation 6.14, which is adaptive according to the average fitness value $\overline{\mu}(t)$ at generation t of the entire population. Equation 6.14 outperforms other mutation rates that depend on external parameters like the fixed mutation rate of 3% per chromosome (see Section 6.3.5). Besides this, the term $Length(S)$, that is guiding to longer snakes, was tested alone and in conjunction with some penalty terms like $\#Lazy$ and $\#Isolated$, where Equation 6.7 maybe one of the most prominent in finding longer snakes in 4-dimensional hypercubes (see Table 6.19 in Section 6.2.3 where seven fitness functions have been proposed).

9.3 The Selection Pressure and Crossover and Mutation Relationship

Hypothesis 3 basically tested the Schema Theorem, which usually is not considered in the design of GAs. It is a common practice to solve problems with GAs using a probability of crossover $p_c = 0.70$ and a probability of mutation of $p_m = 0.001$ without considering the problem itself (this dissertation investigated these values as many others to determine how they influence the performance of GAs), as suggested by Holland (1992). If results are not as expected, changes in both parameters begin to occur and usually the ones that give better results are the ones that are reported. Therefore, basically the principal justification for specific values or a range of values used, in solving a problem with GAs, is the empirical studies. The third general hypothesis proposes then a relationship between the values of the probability of crossover p_c and the probability of mutation p_m that depends on the selection pressure S_p and the schema under consideration (i.e., long schema where $\epsilon = 1$, and small schema where $\epsilon = 1/(l-1)$ with l the length of the chromosome). Such a relationship has been derived from the Schema Theorem proposed by Holland (1992), taking into account the hypothesis that the number of each schema is expected to grow proportionally to its fitness value, considering also non-disruptive factors due to the crossover and mutation operators. This relationship is important, because it shows that a change in one factor (for instance a change in p_m) should take into account a corresponding change in the other(s) (for instance p_c and/or S_p). Besides the interdependency between p_c and p_m, the relationship proposed as in Equations 5.30 and 5.31 highlights the importance of p_m that sometimes is not given in GAs (Piszcz and Soule, 2006b).

The validation of specific hypotheses 3.1 and 3.2 showed that the proposed relationship between S_p, p_c and p_m, as in Equation 7.5, can lead the cases of the snake-in-the-box problem for the 8-dimensional hypercube and the one-max function to obtain on average better quality solutions, where, for example, obeying the Schema Theorem Trade-off (as this relationship is called), longer snakes can be found (see Section 6.3.3). But, if a wrong combination of them is chosen, that can lead the algorithm to mislead the solution or to expend a lot of computational resources to find it (see the exploratory study in Sections 6.3.2 and 6.3.6 and the validation of the specific hypotheses in Sections 7.3.1 and 7.3.2).

The optimization of parameters for GAs is a difficult problem, and there is no a rule or general conclusion that can be applied to every kind of problem (Eiben et al., 1999). However, the theoretical analysis, the empirical studies conducted, and the validation of hypotheses 2 and 3, and the rejection of hypothesis 1, presented in Chapter 7, in this dissertation, could be another piece of the block that could help in the building of parameter setting for GAs. Future research in areas like the relationship between entropy and Hamming diversity; diversity and selection pressure; mutation/crossover probability and diversity; selection pressure, diversity and mutation/crossover probability; a definition of schema diversity, the use of new operators (like the union operator used in Section 6.2.1) that could help fitness functions like Equation 7.3 to avoid false negatives; the beginning of the algorithm with an initial population totally zeroed, so the algorithm has to build the solution from scratch; the possible importance of the mutation operator in the building of better solutions where the crossover operator is quite destructive (as the snake-in-the-box problem presented in Section 6.3.3); and the possible adaptation of S_p, p_c and p_m according to the Schema Theorem Trade-off (see Section 5.2.1) as the GA runs, are some of the areas for continuing this research.

Bibliography

Anderson, J. P., 1980: Computer security threat monitoring and surveillance. Technical report, Fort Washington, PA.

Bace, R. G., 2000: *Intrusion Detection*. MacMillan Technical Publishing, USA.

Bäck, T., 1996: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press.

Bäck, T., A. Eiben, and V. der Vaart, 2000: An empirical study on GAs "without parameters". *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, 315–324.

Bitterman, D. S., 2004: New lower bounds for the snake-in-the-box problem: A prolog genetic algorithm and heuristic search approach. Master Thesis accessed Jun. 2007. URL `http://www.cs.uga.edu/p̃otter/CompIntell/bitterman_derrick_s_200412_ms.pdf`

Burke, E. K., S. Gustafson, and G. Kendall, 2004: Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, **8**, 47–62.

Casella, D. A. and W. D. Potter, 2004: New lower bounds for the snake–in–the–box problem: Using evolutionary techniques to hunt for snakes. *Proceedings of the Florida Artificial Intelligence Research Society Conference*, 264–268.

Cheng, C. D. and A. Kosorukoff, 2004: *Interative One-Max Problem Allows to compare the Performance of Interactive and Human-Based genetic Algorithms*. Chapter in Lecture Notes in Computer Science.

Coello, C. A. C., 1998: A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, **1**, 269–308.

Cohen, P. R., 1995: *Empirical Methods for Artificial Intelligence*. The MIT Press.

Costa, J. C., R. Tavares, and A. Rosa, 1999: An experimental study on dynamic random variation of population size. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 607–612.

Crosbie, M. and G. Spafford, 1995: Applying genetic programming to intrusion detection. *Proceedings of the AAAI Fall Symposium*, 1–8.

Deb, K. and S. Jain, 2004: Running performance metrics for evolutionary multi-objective optimization. Technical report, Indian Institute of Technology Kanpur.

Denning, D. E., 1986: An intrusion-detection model. *Proceedings of the IEEE Symposium on Security and Privacy*, 118–131.

Diaz-Gomez, P. A. and D. F. Hougen, 2005a: Analysis of an off-line intrusion detection system: A case study in multi-objective genetic algorithms. *Proceedings of the Florida Artificial Intelligence Research Society Conference*, 822–823.

—, 2005b: Further analysis of an off-line intrusion detection system: An expanded case study in multi-objective genetic algorithms. *Proceedings of The South Central Information Security Symposium*.

—, 2005c: Improved off-line intrusion detection using a genetic algorithm. *Proceedings of the International Conference on Enterprise Information Systems*, 66–73.

—, 2006a: A genetic algorithm approach for doing misuse detection in audit trail files. *Proceedings of the International Conference on Computing*, 329–335.

—, 2006b: Genetic algorithms for hunting snakes in hypercubes: Fitness function analysis and open questions. *Proceedings of the International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 389–394.

—, 2006c: The snake in the box problem. mathematical conjecture and a genetic algorithm approach. *Proceedings of the Genetic and Evolutionary Computation Conference*, 1409–1410.

—, 2006d: The snake in the box problem: Mathematical conjecture and a genetic algorithm approach. *Proceedings of the Genetic and Evolutionary Computation Conference*, 1409–1410.

—, 2006e: Three approaches to intrusion detection: Analysis and enhancements. *Proceedings of the National Computer and Information Security Conference*.

—, 2007a: Empirical study: Initial population diversity and genetic algorithm performance. *In Proceedings of the Conference on Artificial Intelligence and Pattern recognition*.

—, 2007b: Hunting for snakes in hypercubes using genetic algorithms. *International Journal of Computer & Information Science*, to appear.

—, 2007c: Initial population for genetics algorithms: A metric approach. *Proceedings of the International Conference on Genetic and Evolutionary Methods*.

—, 2007d: MISUSE DETECTION: A neural network vs. a genetic algorithm approach. *Proceedings of the International Conference on Enterprise Information Systems*.

—, 2007e: MISUSE DETECTION: An iterative process vs. a genetic algorithm approach. *Proceedings of the International Conference on Enterprise Information Systems*.

—, 2007f: A trade-off of the schema theorem. *In Proceedings of the Conference on Artificial Intelligence and Pattern recognition*.

Efron, B. and R. J. Tibshirani, 1993: *An Introduction to the Bootstrap*. Chapman & Hall/CRC.

Eiben, A. E., R. Hinterding, and Z. Michalewicz, 1999: Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, **3**, 124–141.

Frederick, W. G., R. L. Sedlmeyer, and C. M. White, 1993: The hamming metric in genetic algorithms and its application to two network problems. *Proceedings of the ACM/SIGAPP Symposium on Applied Computing*, 126–130.

Fried, D. and M. Zissman, 1998: Intrusion detection evaluation. Technical report, Lincoln Laboratory, MIT, accessed March 2004.
URL `http://www.ll.mit.edu/IST/ideval/`

Fu, X. and L. Wang, 2005: A rule extraction system with class-dependent features. *Evolutionar Computation in Data Mining*, 79–99.

Giguere, P. and D. E. Goldberg, 1998: Population sizing for optimum sampling with genetic algorithms: A case study of the onemax problem. *Proceedings of the Third Anual Genetic Programming Conference*.

Goldberg, D. E. and K. Sastry, 2001: A practical schema theorem for genetic algorithm design and tuning. *Proceedings of the Genetic and Evolutionary Computation Conference*, 328–335.

Greenberg, D. S. and S. N. Bhatt, 1990: Routing multiple paths in hypercubes. *Proceedings of the Second Annual ACM Symposium on parallel Algorithms and Architectures*.

Grefenstette, J. J., 1986: Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-16**, 122–128.

Ham, F. M. and I. Kostanic, 2001: *Principles of Neurocomputing for Science & Engineering*. Mc Graw Hill.

Harik, G. R. and F. G. Lobo, 1999: A parameter-less genetic algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference*, 258–265.

Holland, J., 1992: *Adaptation in Natural and Artificial Systems*. MIT Press.

Jaccard, J., 1998: Interaction effects in factorial analysis of variance. a SAGE UNIVERSITY PAPER, monograf Series/Number 07-118.

Jaroslaw Arabas, Z. M. and J. Mulawka, 1995: GAVaPS—a genetic algorithm with varying population size. *Proceedings of the IEEE International Conference on Evolutionary Computation*, 73–78.

Klee, V., 1970: What is the maximum length of a d-Ddimensional snake. *American Mathematics Monthly*, **77**, 63–65.

Koumousis, V. K. and C. P. Katsaras, 2006: A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation*, **10**, 19–28.

Lakshmivarahan, S. and S. Dhall, 1990: *Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problem*. MacGraw-Hill.

Li, W., 2003: A genetic algorithm approach to network intrusion detection. Global Information Assurance Certification. Practical assignment, accessed July 19/2004.
URL `http://www.giac.org/practical/GSEC/Wei_Li_GSEC.pdf`

Lobo, F. G. and D. E. Goldberg, 2004: The parameter-less genetic algorithm in practice. *Information Sciences—Informatics and Computer Science*, **167**, 217–232.

Lobo, F. G. and C. F. Lima, 2005: A review of adaptive population sizing schemes in genetic algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference*, 228–234.

Lunacek, M. and D. Whitley, 2006: The dispersion metric and the CMA evolution strategy. *Proceedings of the Genetic and Evolutionary Computation Conference*, 447–484.

Mann, P. S., 2007: *Introductory Statistics*. Wiley.

McPhee, N. E. and N. Hopper, 1999: Analysis of genetic diversity through population history. *Proceedings of the Genetic and Evolutionary Computation Conference*, 1112–1120.

Mé, L., 1993: Security audit trail analysis using genetic algorithms. *Proceedings of the International Conference on Computer safety, reliability, and Security*, 329–340.

—, 1998: GASSATA, a genetic algorithm as an alternative tool for security audit trail analysis. *Proceedings of the First International Workshop on the Recent Advances in Intrusion Detection*.

Mitchell, M., 1998: *An Introduction to Genetic Algorithms*. MIT Press.

Pelikan, M., D. E. Goldberg, and E. Cantú-Paz, 2000: Bayesian optimization algorithm, population sizing, and time to convergence. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois.

Piszcz, A. and T. Soule, 2006a: Genetic programming: Optimal population sizes for varying complexity problems. *Proceedings of the Genetic and Evolutionary Computation Conference*, 953–954.

—, 2006b: A survey of mutation techniques in genetic programming. *Proceedings of the Genetic and Evolutionary Computation Conference*, 951–952.

Potter, W. D., R. W. Robinson, J. A. Miller, K. Kochut, and D. Z. Redys, 1994: Using the genetic algorithm to find snake-in-the-box codes. *Proceedings of the 7th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, 421–426.

Rajan, D. S. and A. M. Shende, 1999: Maximal and reversible snakes in hypercubes. *Proceedings of the Australasian Conference on Combinatorial Mathematics and Combinatorial Computing*.

Reeves, C. R., 1993: Using genetic algorithms with small populations. *Proceedings of the 5th International Conference on Genetic Algorithms*, 92–99.

Rosca, J. P., 1995: Entropy-driven adaptive representation. *Proceedings of the Workshop Genetic Programming: From Theory to Real-World Applications*, 23–32.

Schneier, B., 2000: *Secrets & Lies: Digital Security in a Networked World*. Wiley Computer Publishing.

Shannon, C. E., 1948: A mathematical theory of communication. *The Bell System Technical Journal*, **27**, 379–423, 623–656.

Snevily, H. S., 1994: The Snake-in-the-Box problem: A new upper bound. *Discrete Mathematics*, 307–314.

Tjaden, B. C., 2004: *Fundamentals of Secure Computer Systems*. Franklin and Beedle & Associates.

Weisstein, E. W., 2006: Snake – from mathworld. Accessed May 2007.
URL `http://mathworld.wolfram.com/Snake.html`

Yu, T.-L., K. Sastry, D. E. Goldberg, and M. Pelikan, 2006: Population sizing for entropy-based model building in genetic algorithms. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois.

Yu, T.-L., K. Sastry, D. E. Goldberg, and K. Sastry, 2003: Optimal sampling and speed-up for genetic algorithms on the sampled onemax problem. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois.

Zitzler, E., K. Deb, and L. Thiele, 2000: Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, **8**, 173–195.